

edited by Aleksander Gurlo

Anthony Yu-Tung Wang

Paying Attention to Materials: Transformers in the Context of Materials Informatics



Paying Attention to Materials: Transformers in the Context of Materials Informatics

vorgelegt von
M.Sc. RWTH, BAsC.
Anthony Yu-Tung Wang
ORCID: 0000-0002-7947-0309

von der Fakultät III – Prozesswissenschaften –
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzende: Prof. Dr.-Ing. Claudia Fleck
Gutachter: Prof. Dr. Aleksander Gurlo
Gutachter: Prof. Dr. Taylor D. Sparks
Gutachter: Dr. Michael W. Gaultois

Tag der wissenschaftlichen Aussprache: 9. Dezember 2021

Berlin 2022

The scientific series *Advanced Ceramic Materials* of the Technische Universität Berlin is edited by: Prof. Dr. Aleksander Gurlo.

This manuscript is protected by copyright.

Cover image: David Karl | Segerkegel | 2017 | CC BY 4.0
<https://creativecommons.org/licenses/by/4.0/>

Layout/Typesetting: Anthony Yu-Tung Wang

Published online on the institutional repository of the Technische Universität Berlin:
DOI 10.14279/depositonce-14888
<https://dx.doi.org/10.14279/depositonce-14888>

"All models are wrong, but some are useful."
— George E. P. Box (1987)

Abstract

The fast and affordable development of novel materials is needed in order to enable technological advancements in application areas such as clean energy, healthcare, sustainable transport, and climate-friendly consumption. However, the development of novel materials is not a trivial task. One of the biggest challenges in materials design and discovery is the enormous search space of possible material compositions available, also referred to as the “chemical whitespace”.

Faced with the high risk, high reward nature of materials exploration, materials scientists have increasingly moved away from traditional trial and error methods and instead adapted new data-driven methods of materials discovery. The rapid development of data science, machine learning (ML), and deep learning (DL) as well as the influx of high-quality materials property datasets have led to the development of the new field of materials informatics (MI). This new paradigm has drastically changed the way in which materials are understood, predicted, discovered, and designed.

Despite promising developments in this relatively young field, there are several open issues that need to be addressed. The lack of guidelines and established procedures to ensure high quality research in MI impedes the pace of further development in this field. Furthermore, the current techniques for representing and modeling chemical compositions are flawed and unsuitable to be used in the search of novel materials. Lastly, the prevalence of black-box DL models without model interpretability limits the trust and adoption of these models in academia and industry. Accordingly, the main aims of this work are (1) to propose a set of best practices and protocols for conducting and reporting MI studies, and (2) to improve the state of the art in materials property predictions by introducing interpretable DL techniques for representing and modeling chemical compounds.

In the first work described in this thesis, the fundamental ideas and considerations of using data-driven methods for materials science are introduced. A broad set of guidelines and protocols for ensuring the reliable, reproducible, and comparable reporting

of research results in MI studies is established. Common software tools, methodologies, and materials data repositories are presented. Lastly, the full procedure of an ML study including data processing, feature engineering, model training, evaluation and comparison is demonstrated using the prediction of heat capacity for solid inorganic compounds as an example.

In the second work, a novel DL model named “Compositionally Restricted Attention-Based network (CrabNet)”, based on the Transformer self-attention mechanism, is introduced. CrabNet is benchmarked on 28 materials property datasets and is shown to match or exceed state-of-the-art models in the prediction of inorganic material properties. The benefits of learning element-element interactions within chemical compounds using the self-attention mechanism are discussed. Furthermore, a new way of representing chemical composition which overcomes some of the limitations present in current techniques is developed. Lastly, the opportunities to study model interpretability methods in CrabNet are previewed.

Continuing in the third work described in this thesis, the model interpretability of CrabNet is further examined. Intrinsic model interpretability methods are added to CrabNet and used to extract additional information about the model and data representations during the modeling process. The extracted information is processed and visualized into static and interactive figures as well as video animations. The examination of these visualizations and additional information reveals well-known chemical patterns about the elements and compounds, intuitively suggesting that CrabNet is able to learn the element properties, element interactions, and how they together dictate materials properties. Furthermore, the dataset quality as well as the self-attention mechanism are also discussed for their significance towards an improved and interpretable modeling of materials properties. Lastly, the potential benefits of applying interpretable modeling methods in academia and industry are discussed.

Overall, the methods, results, and considerations discussed in this dissertation are presented in a way to educate and empower interested materials science researchers to undertake their own materials informatics research.

Kurzfassung

Die schnelle und erschwingliche Entwicklung neuartiger Materialien wird benötigt, um technologische Fortschritte in Anwendungsbereichen wie der sauberen Energie, dem Gesundheitswesen, dem nachhaltiger Verkehr und dem klimafreundlichen Konsum zu ermöglichen. Die Entwicklung neuartiger Materialien ist jedoch keine triviale Aufgabe. Eine der größten Herausforderungen bei Materialdesign und -entdeckung ist der enorme Suchraum möglicher Materialzusammensetzungen, der auch als „chemical whitespace“ bezeichnet wird.

Angesichts der risikointensiven aber aussichtsreichen Natur der Materialerforschung ziehen Materialwissenschaftler zunehmend von herkömmlichen Versuchs-und-Irrtums-Methoden weg und adaptieren stattdessen neue datengetriebene Methoden der Materialentdeckung. Die schnelle Entwicklung von Data Science, maschinellem Lernen (ML) und Deep Learning (DL) sowie der Zustrom hochwertiger Materialdatensätze haben zu der Entwicklung des neuen Gebiets der Materialinformatik (MI) geführt. Dieses neue Paradigma hat die Art und Weise mit der Materialien verstanden, vorhergesagt, entdeckt und entworfen werden drastisch verändert.

Trotz vielversprechender Entwicklungen in diesem relativ neuen Bereich gibt es mehrere offene Probleme, die behoben werden müssen. Der Mangel an Richtlinien und etablierten Verfahren zur Gewährleistung hochwertiger Forschung in der MI behindert das Tempo der Weiterentwicklung in diesem Bereich. Darüber hinaus sind die derzeitigen Techniken zur Darstellung und Modellierung chemischer Zusammensetzungen fehlerhaft und ungeeignet, um bei der Suche nach neuartigen Materialien verwendet zu werden. Schließlich begrenzt die Prävalenz von Black-Box-DL-Modellen ohne Modellinterpretierbarkeit das Vertrauen und die Akzeptanz dieser Modelle in der Wissenschaft und Industrie. Dementsprechend sind die Hauptziele dieser Arbeit (1) einen Satz bewährter Verfahren und Protokolle zur Durchführung und Berichterstellung von MI-Studien vorzuschlagen und (2) den Stand der Technik in der Vorhersage

von Materialeigenschaften durch die Einführung von interpretierbaren DL-Techniken zur Darstellung und Modellierung chemischer Verbindungen zu verbessern.

In der ersten in dieser Dissertation beschriebenen Arbeit werden die grundlegenden Ideen und Überlegungen zur Verwendung von datengetriebenen Methoden für die Materialwissenschaft eingeführt. Eine breite Reihe von Richtlinien und Protokollen, um die zuverlässige, reproduzierbare und vergleichbare Berichterstattung von Forschungsergebnissen in ML-Studien zu gewährleisten, wird etabliert. Gängige Software-Tools, Methoden und Repositorien für Materialdaten werden dargestellt. Schließlich wird das vollständige Verfahren einer ML-Studie für die Vorhersage der Wärmekapazität fester anorganischer Verbindungen, einschließlich Datenverarbeitung, Feature Engineering, Modelltraining, -auswertung und -vergleich, als Beispiel gezeigt.

In der zweiten Arbeit wird ein neuartiges DL-Modell namens „Compositionally Restricted Attention-Based network“ (CrabNet), basierend auf dem Transformer Self-Attention-Mechanismus, eingeführt. CrabNet wird anhand von 28 Materialdatensätzen evaluiert und kann den Stand der Technik bei der Vorhersage anorganischer Materialeigenschaften erreichen oder übertreffen. Die Vorteile, Wechselwirkungen zwischen Elementen in chemischen Verbindungen unter Verwendung des Self-Attention Mechanismus zu lernen, werden ebenfalls diskutiert. Darüber hinaus wird eine neue Art der Repräsentation für chemische Zusammensetzungen, die einige der in aktuellen Techniken vorhandenen Einschränkungen überwindet, präsentiert. Schließlich wird eine Vorschau der Möglichkeiten Modellinterpretationsmethoden in CrabNet zu untersuchen gezeigt.

Fortgesetzt in der dritten Arbeit wird die Modellinterpretierbarkeit von CrabNet weiter untersucht. Intrinsische Modellinterpretationsmethoden werden zu CrabNet hinzugefügt und verwendet, um zusätzliche Informationen über das Modell und die Datenrepräsentationen während des Modellierungsprozesses zu extrahieren. Die extrahierten Informationen werden in statischen und interaktiven Abbildungen sowie Videoanimationen verarbeitet und visualisiert. Die Untersuchung dieser Visualisierungen und der zusätzlichen Informationen ergibt bekannte chemische Muster hinsichtlich der Elemente und deren Verbindungen, die intuitiv darauf hindeuten, dass CrabNet die Elementeigenschaften, -wechselwirkungen und deren Einfluss auf die Materialeigenschaften, lernen kann. Darüber hinaus werden die Qualität des Datensatzes und der

Self-Attention-Mechanismus hinsichtlich ihrer Bedeutung für eine verbesserte und interpretierbare Modellierung von Materialeigenschaften diskutiert. Schließlich werden die potenziellen Vorteile der Anwendung interpretierbarer Modellierungsmethoden in der Wissenschaft und der Industrie diskutiert.

Insgesamt sind die in dieser Dissertation diskutierten Methoden, Ergebnisse und Erwägungen so dargestellt, dass sie interessierte Materialwissenschaftler dazu ermächtigen sich auf diesem Gebiet weiterzubilden, um ihre eigene Forschung in der Materialinformatik durchzuführen.

Table of contents

List of figures	ix
List of tables	xi
1 Introduction	1
1.1 Structure of the thesis	1
1.2 Motivation and goals	3
2 Humanity’s material challenges	7
2.1 Chemical whitespace and the pace of materials research	8
2.2 Materials informatics: a new paradigm	12
2.3 Machine learning for materials science	14
3 Best practices for machine learning in materials science	17
3.1 The need for best practices in materials informatics	17
3.2 Publication 1: Machine Learning for Materials Scientists: An Introductory Guide toward Best Practices	19
4 Novel attention-based learning of materials properties	113
4.1 Lack of adequate composition featurization techniques	113
4.2 Lack of structure-agnostic deep learning	115
4.3 Transformers and the self-attention mechanism	116
4.4 Publication 2: Compositionally restricted attention-based network for materials property predictions	119
5 Interpretable deep learning with CrabNet	137
5.1 Lack of interpretable modeling in materials science	137
5.2 Publication 3: CrabNet for Explainable Deep Learning in Materials Science: Bridging the Gap Between Academia and Industry	139
6 Summary and outlook	163
7 References	167
List of abbreviations	177
Acknowledgements	179

List of figures

2.1	The combinatorial explosion resulting from the number of possible element combinations in an n -element compound renders the brute-force exploration of chemical whitespace impossible. Additional stoichiometric complexity and their granularity—dopants are an extreme example—further exacerbates this already impossible task	9
2.2	The accelerating pace of materials science research as measured by the number of new articles published per year for the period from 1940 to 2020, grouped by the topic keywords as shown. Data collected from the Web of Science. Figure adapted with permission from [31]	11
2.3	The four paradigms of materials science. In the traditional approach (left), new materials are discovered by experimentation, theory, or computation (also referred to as the 1 st , 2 nd , and 3 rd paradigms). In the 4 th paradigm (right), all of the available materials data is collected in accessible data infrastructures, and machine learning and data science approaches are used to discover new materials in the paradigm of data-driven materials science	12
2.4	A typical workflow for materials discovery, integrating materials informatics, machine learning, simulation and experimentation to narrow down a large number of potential candidate materials to a few promising candidates. Figure adapted with permission from [51]	15
4.1	Example of the composition-based feature vector (CBFV) featurization of Al_2O_3 . The compound is featurized using the individual element vectors of aluminum and oxygen (ν_{Al} and ν_{O}), which are first weighted by the fractional prevalence of the elements. The featurized vector $\nu_{\text{Al}_2\text{O}_3}$ of Al_2O_3 is obtained by calculating the descriptive statistics (<i>e.g.</i> , mean, variance, and range) of the weighted element vectors. Another example featurized vector ν_{SiC} of the compound SiC is also shown	114

List of tables

4.1	Examples of common composition-based feature vector (CBFV) feature sets, including their source and requirements on domain knowledge and hand-engineering	113
-----	---------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----

1 Introduction

1.1 Structure of the thesis

The present work is a compilation of three peer-reviewed journal publications, together covering the results obtained within this cumulative dissertation:

- 1) A. Y.-T. Wang, R. J. Murdock, S. K. Kauwe, A. O. Oliynyk, A. Gurlo, J. Brgoch, K. A. Persson, and T. D. Sparks. Machine Learning for Materials Scientists: An Introductory Guide toward Best Practices, *Chemistry of Materials*, **2020**, 32 (12): 4954–4965.

DOI: 10.1021/acs.chemmater.0c01907.

The published manuscript is reproduced in Chapter 3.2 of the dissertation.

- 2) A. Y.-T. Wang, S. K. Kauwe, R. J. Murdock, and T. D. Sparks. Compositionally restricted attention-based network for materials property predictions, *npj Computational Materials*, **2021**, 7: 77.

DOI: 10.1038/s41524-021-00545-1.

The published manuscript is reproduced in Chapter 4.4 of the dissertation.

- 3) A. Y.-T. Wang, M. S. Mahmoud, M. Czasny, and A. Gurlo. CrabNet for Explainable Deep Learning in Materials Science: Bridging the Gap Between Academia and Industry, *Integrating Materials and Manufacturing Innovation*, **2022**.

DOI: 10.1007/s40192-021-00247-y.

The published manuscript is reproduced in Chapter 5.2 of the dissertation.

In addition to the works listed above, I co-authored one publication closely related to the topic of this thesis:

R. J. Murdock, S. K. Kauwe, A. Y.-T. Wang, and T. D. Sparks. Is Domain Knowledge Necessary for Machine Learning Materials Properties?, *Integrating Materials and Manufacturing Innovation*, **2020**, 9 (3): 221–227.

DOI: 10.1007/s40192-020-00179-z.

1 INTRODUCTION

This dissertation is organized as follows:

Chapter 1 describes the motivation, the **three goals** of this Ph.D. work, and the approach taken to address the goals.

Chapter 2 introduces the reader to the background of machine learning and deep learning in the context of materials science. The new paradigm of using materials informatics for conducting materials design and discovery is also introduced. In addition, a brief overview of the notable works in this field are presented, including the open research questions leading up to this work.

Chapter 3 emphasizes the need for a set of fundamental guidelines and best practices to ensure reproducible, comparable, and credible reporting of research results in materials informatics studies involving data science or machine learning. The first peer-reviewed publication that addresses the first goal of this thesis is also presented.

Chapter 4 highlights the advantages of the Transformer attention-based deep learning of materials properties that is used by the Compositionally Restricted Attention-Based network (CrabNet) developed as part of this dissertation. The unique features of CrabNet and its underlying composition featurization method are then compared against other common methods in the literature. The results and benefits of implementing attention-based methods for the learning of materials properties are presented in the second peer-reviewed publication that addresses the second goal of this thesis.

Chapter 5 extends CrabNet by introducing additional model interpretability methods that aim to improve the understanding of the model, the modeling process as well as the learned chemical interactions underlying materials properties. The results of extending model interpretability in CrabNet and their potential broader impact in increasing the adoption of deep learning methods in materials science are discussed in the third peer-reviewed publication that addresses the third goal of this thesis.

Chapter 6 summarizes the outcomes of this dissertation work and suggests future research directions for the applications of Transformers, attention-based deep learning, and interpretable models in materials informatics.

Please note that the cited references in this dissertation are displayed at the end of the thesis (page 167). Each of the publications contain their own references, which are shown at the end of each manuscript.

1.2 Motivation and goals

Ever since prehistoric times, humankind has tirelessly worked on the understanding and improvement of materials properties. New discoveries and developments of materials with improved or unprecedented properties have profoundly changed important aspects of past civilizations such as culture, agriculture, architecture, and even warfare. The changes are so drastic such that entire archaeological periods of time—the Stone, Bronze, Iron, and Silicon Ages—were named after the prominent materials used.

The necessity to discover and develop new engineering materials persists today, driven by the need to meet society's greatest challenges in ensuring human and environmental well-being while retaining economic prosperity. However, the simultaneous requirements of developing novel materials quickly and affordably present many challenges.

Many common engineering materials we know today were discovered as a result of scientific serendipity: on these occasions, chance played a significant role in advancing materials science. But more often than not, new materials are discovered through incremental improvements via trial and error experimentation. However, the exploration of the materials design space is by no means trivial. The complexities of the interactions between the material composition, structure, processing and the material properties serve to make the understanding and modeling of materials a difficult task. The enormous combinatorial space resulting from the large number of possible elements and stoichiometries—the number of combinatorially possible materials has been estimated to be as high as a googol (10^{100})—further complicates this issue.

Fortunately, the vast amounts of experimental and simulated materials property data accumulated through decades of research presented an ideal opportunity for a paradigm shift towards the data-driven study of materials science—also known as “materials informatics” (MI). Crucial to this revolution is also the development of computational and statistical methods such as data science and artificial intelligence (AI). In particular, machine learning (ML) and deep learning (DL) methods have been successfully applied for the classification and prediction of materials properties, enabling groundbreaking materials design and discovery workflows that were previously impossible.

Despite the promising results of early works in this relatively new field, there remain several critical and open questions to be addressed in the adaptation of ML and DL

1 INTRODUCTION

methods to the materials science domain. The ease of use and access of publicly available data science tools is a double-edged sword that sometimes misguides materials scientists who foray into the field of materials informatics. The inexperience of materials scientists in conducting and documenting studies involving data science, ML and DL often leads to misleading or incorrect results.

Furthermore, while methods of featurizing materials compositions into input data for the ML and DL models exist and are widely used to date, these methods make fundamentally incorrect assumptions about the underlying material systems. These techniques work well for lower-accuracy ML predictions and for cases where data volume is limited; however, the fundamental assumptions severely limit their usefulness and relevance when developing strong, accurate and robust DL models using large materials property datasets.

Additionally, there is a lack of models that can adequately capture and model the interactions between constituent elements in a compound while simultaneously not requiring crystal structure information—an important prerequisite for the discovery of novel materials.

Lastly, while there is large academic interest in embracing data-driven methods for the study of materials, the adoption of these methods in industry remains low due to the lack of model transparency and model interpretability techniques. This is further exacerbated by the lack of intuitive understanding of modeling and decision-making processes, leading to low overall trust and adoption.

This thesis aims to address the aforementioned issues in three topics: (1) the establishment of best practices for current and future researchers interested in conducting materials informatics research, (2) the introduction of novel attention-based modeling and chemical composition featurization techniques for materials property prediction, and (3) the introduction of model interpretability using the models developed as part of this thesis work.

The **first goal** of this work is to establish a set of best practice methods and protocols for materials scientists who are interested in performing machine learning or deep learning research in the context of materials informatics. To this aim, guidelines and best practices are proposed regarding the obtaining and treatment of data, feature

engineering, model training, validation, evaluation and comparison, model and architecture sharing, and finally publication. A selection of studies employing materials informatics for the prediction of materials properties for a variety of applications is also reviewed. Furthermore, popular materials data repositories are highlighted. Python code to demonstrate the concepts and perform an example machine learning workflow is provided as interactive Jupyter notebooks. The applicability and robustness of materials informatics methods for materials science problems are also discussed. Together, the set of guidelines and best practices ensures that reported results from researchers are reproducible and comparable with those from other researchers, and will improve the overall research and manuscript quality in the field.

The **second goal** of this work is to introduce a new, structure-agnostic deep learning model to learn and predict materials properties, while simultaneously improving the state of the art in model performance. To this end, a novel model based on the Transformer self-attention mechanism is introduced and benchmarked against other common models on material property regression tasks using 28 benchmark datasets. Furthermore, a featurization technique is developed to enable the attention modeling which preserves element identity within the chemical compounds while not ignoring trace elements. Additionally, the opportunities to explore model interpretability methods in the newly-introduced Compositionally Restricted Attention-Based network (CrabNet) are discussed.

The **third goal** of this work is to demonstrate that model interpretability methods can be built into CrabNet and that these methods can lead to additional understanding and intuition about the chemical behaviour in elements and compounds. Furthermore, by examining the model training process, potential modeling or dataset errors can be discovered, which may highlight further insights leading to a better understanding of the phenomena governing materials properties. Lastly, the introduction of interpretable methods to black-box models will lead to wider acceptance and adoption in industry and academia alike.

2 Humanity's material challenges

Traditionally, the search for new or alternative materials with desired properties or behaviours has been a slow and difficult task. Materials scientists often dream of designing or finding completely new materials with better performance at a lower cost compared to existing materials. However, this dream is often only partially realized: instead of discovering novel materials by pure ingenuity and hard work, scientists discover new materials only through incremental improvements in empirical, theoretical, or computational research.

“Genius is one percent inspiration and ninety-nine percent perspiration”, according to the famed American inventor Thomas Edison, who tested more than 6000 materials to find a suitable candidate for the filament in an incandescent light bulb [5, 6]. Typically, the discovery of new materials arises out of a combination of solid scientific understanding, rigorous experimentation, and the unwavering dedication to persevere after numerous failures. Occasionally, a stroke of luck (commonly referred to as “serendipity”) is also involved in the process.

Many of the world's crucial materials were discovered through a fortunate mixture of pure chance, ingenuity, and unbiased curiosity of the scientist. Shatterproof glass, dynamite, Viagra, penicillin, quinine, insulin, artificial dyes, super glue, vulcanized rubber, synthetic plastics including Teflon, Velcro, artificial sweeteners, shape memory alloys and many other well-known materials were all discovered in an “Eureka” moment due to the combination of a happy accident and an observant scientist [7–17].

This type of fortuitous materials discovery through serendipity is without doubt a major contributor to scientific progress. However, the rapidly changing society and technology progress has placed increasing demands on materials development and innovation. Unfortunately, serendipities are rare and cannot be relied upon to address the ever-changing demands and material requirements: out of the fourteen Grand Challenges of Engineering in the 21st Century as identified by the National Academy

2 HUMANITY'S MATERIAL CHALLENGES

of Engineering, eight of the challenges will require the discovery and development of novel materials [18]:

- Make solar energy affordable
- Engineer better medicines
- Restore and improve urban infrastructure
- Provide access to clean water
- Provide energy from fusion
- Prevent nuclear terror
- Manage the nitrogen cycle
- Develop carbon sequestration methods

Many excellent researchers around the world are certainly working tirelessly to meet these challenges; however, the search for novel materials is not well-defined and is not by any means a simple task. Considering the well-studied and -documented periodic table of the elements as well as the wealth of materials science knowledge available in the literature and in the collective scientific community, it may seem trivial to combine the right expertise, theory, and praxis to design a new material which can deliver a desired property. However, materials discovery is anything but straightforward, and there remains one major hurdle when faced with the task of finding the next novel material: *where do we start?*

2.1 Chemical whitespace and the pace of materials research

Indeed, one of the most significant hurdles for the discovery of novel or alternative engineering materials is the massive number of possible element combinations in a given chemical compound. This space of all possible stoichiometric combinations is also known as the “chemical whitespace” [19]. The synthesis of novel materials requires a large amount of dedication and trial and error processes to find the optimal chemical composition, synthesis processes and conditions (think of Edison’s light bulb). Likewise, the characterization of the synthesized materials is also not straightforward at times—techniques such as diffraction, spectroscopy, and electron microscopy can require a large amount of effort and time. Furthermore, some novel materials could contain exotic elements, phases, or structures, which are costly to obtain and/or synthesize.

Figure 2.1 shows the number of possible compounds that can be formed given n unique elements (out of a pool of 80 stable elements). Note that the number of possible ways to combine n elements grows dramatically with n , and surpasses one million when $n = 4$. In other words, for a compound with four elements, there are over one million potential ways to choose and combine the elements together.

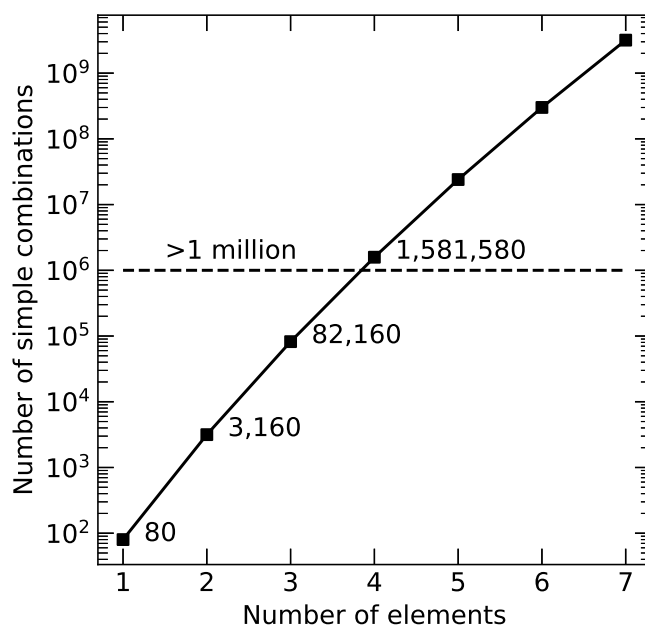


Figure 2.1: The combinatorial explosion resulting from the number of possible element combinations in an n -element compound renders the brute-force exploration of chemical whitespace impossible. Additional stoichiometric complexity and their granularity—dopants are an extreme example—further exacerbates this already impossible task.

Furthermore, these calculations assume that each element is equally abundant in the compound (*i.e.*, for a compound with four elements, each element will have a 25 at% prevalence in the compound). Thus far, the additional dimensions of varying stoichiometry have not been taken into account. If we wish to introduce stoichiometry variances in the search space, then the number of possible combinations becomes exponentially larger. In fact, the number of combinatorially possible materials has been estimated to be as high as a googol (10^{100}), which is more than the total number of atoms in the known universe [20]. Even for simple binary and ternary compounds, it would be an impossible task to synthesize, characterize, and study all of the elemental combinations

in search of a desired material property, given the very large combinatorial space of possible elements, stoichiometries, and the limited amount of research time and funding available [9, 19, 21].

As a consequence of this high-risk, high-reward nature of discovering new materials, many researchers choose instead to focus on incremental material improvements that can be achieved by optimization and exploration within already-known chemical systems. The search for new materials is thus typically limited to educated guesses based on known materials science phenomena or local optimizations based on known material systems. While these methods can lead to successful and positive results, they are “brute-force” trial and error methods for discovering new materials, in the sense that components are progressively swapped out, or added to existing or known compositions, and then tested for improved performance [9, 19, 22–27]. For large materials systems with many elements, this is predictably very costly and time-consuming.

These search methodologies can also fail—for example, if a given system is already as optimized as it can be, no further fine-tuning or swapping out elements can improve it more. Furthermore, some materials phenomena which rely on minute changes in the chemical composition, structure, or other rare events such as dopant effects, can be completely missed. Additionally, it can also be possible that entirely new systems, with new elements or compounds, are missed entirely, since these brute-force methods search only locally near a starting chemical composition (*i.e.*, they are optimizing locally instead of globally across the entire chemical space). This can be a limitation especially for classes of materials for which there are only a small number of known compositions.

Chemical whitespace can be navigated up to a certain degree by a scientist with proper experience, understanding of the latest research, and intuition. Naturally, the publications made by the worldwide scientific community play a key role in the dissemination of the latest research topics and findings in this context. It is possible to imagine that an expert can follow the newest developments by reading all of the latest publications, take inspiration from that, and then identify the next promising topic(s) of research. To this end, Edison argues that “a ‘genius’ is often merely a talented person who has done all of his or her homework”. However, is it actually possible for a modern-day scientist to keep up with the ever-increasing number of publications? In Edison’s time, that may have been possible—but it is certainly unimaginable now. Figure 2.2 shows the number

2.1 CHEMICAL WHITESPACE AND THE PACE OF MATERIALS RESEARCH

of new publications per year, as indexed by the Web of Science for selected topics in materials science. We can observe that the number of scientific publications for all topics has been steadily increasing year-over-year, and that the number of publications doubles approximately every 10 to 15 years [28]. In the year 2020 alone, many materials science topics accumulated over 10^4 new publications! In fact, this growing trend in publications and a similar growth in the number of scientists have been observed since the 1600s and continues to this day—90% of all of the scientists that have ever lived are still alive today [29, 30].

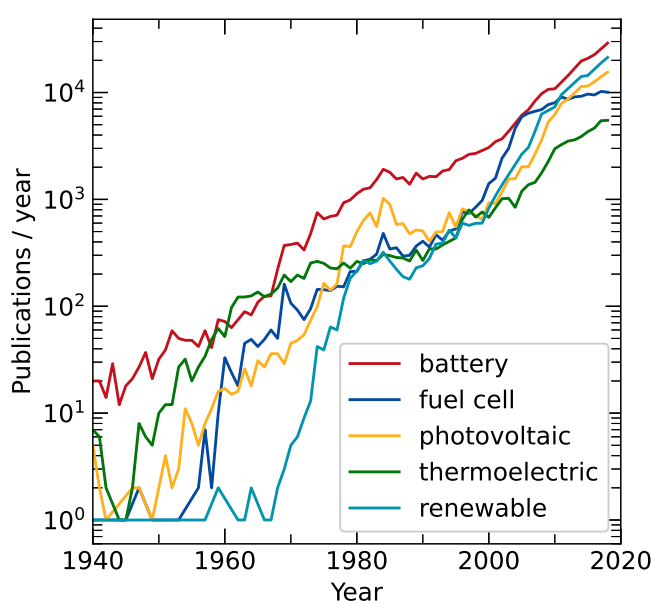


Figure 2.2: The accelerating pace of materials science research as measured by the number of new articles published per year for the period from 1940 to 2020, grouped by the topic keywords as shown. Data collected from the Web of Science. Figure adapted with permission from [31].

It is impossible for any researcher to read tens of thousands of papers every year, and it is just as unfeasible to expect this researcher to manually synthesize and characterize thousands of compounds every year. But does this mean this researcher cannot do his or her homework properly? Maybe not. It is clear that the current approaches for materials discovery require a major overhaul in view of the manual experimental and research methods. There needs to be a structured, reproducible and rational method to explore chemical whitespace, which should not only be economical, but also quick, accurate,

and precise [19]. Materials informatics (MI) combined with machine learning (ML) represents this method, and provides a unique way to integrate scientific knowledge, data, and theory for the data-driven discovery of novel materials. This could be a solution to our researcher's problem.

2.2 Materials informatics: a new paradigm

The development of materials science (and of materials informatics, by extension) closely followed the breakthroughs and advancements of science and technology over the millennia. These can be divided into four major paradigms (Figure 2.3).

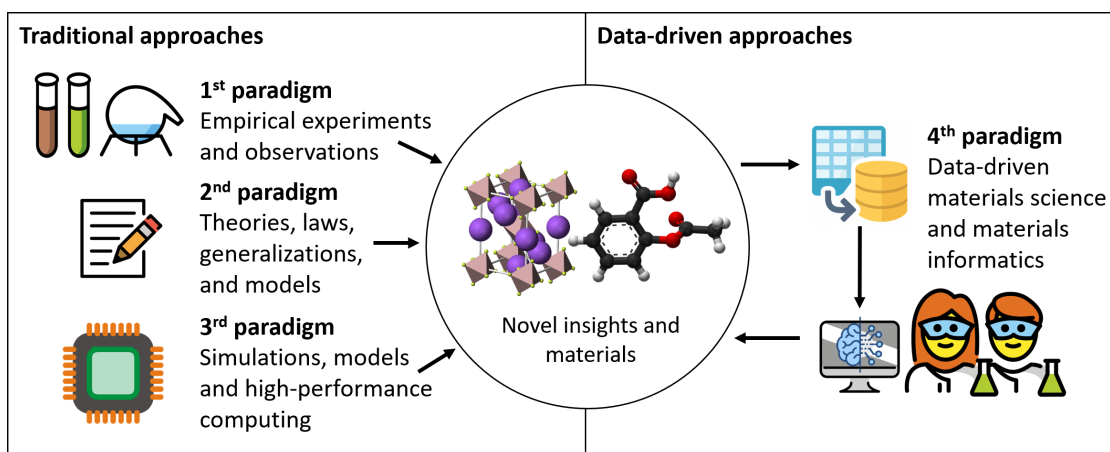


Figure 2.3: The four paradigms of materials science. In the traditional approach (left), new materials are discovered by experimentation, theory, or computation (also referred to as the 1st, 2nd, and 3rd paradigms). In the 4th paradigm (right), all of the available materials data is collected in accessible data infrastructures, and machine learning and data science approaches are used to discover new materials in the paradigm of data-driven materials science.

For a long time in humankind's history, materials science was empirical and followed general observations and experience gathered through experimentation. This represents the first paradigm in materials science, the paradigm of empirical observations. Then came the age of theories and generalizations, with many laws, equations and systems being formulated to explain complex interactions like thermodynamics—the second paradigm of theories and generalizations. Then, when theoretical models became too

2.2 MATERIALS INFORMATICS: A NEW PARADIGM

complex, computational methods were used to simulate the models and to obtain their solutions, in the third paradigm of materials science. Some well-known examples of these first-principles computational methods are molecular dynamics (MD) and density functional theory (DFT). Computational methods now permeate almost all scientific fields and are used to solve a large variety of problems.

With the constant improvement of models and high-throughput computation, the amount of data generated by model-based research—together with data from empirical experiments—have led to the fourth paradigm in materials science in the last few years. This is the (big-)data-driven study of materials science [32]. Data-driven materials science combines elements from the first three paradigms (experimentation, observation, theory, and computation) and represents a new, disruptive paradigm in which the vast amounts of currently-available materials data can be explored to draw interesting and novel insights [32–34].

While data-driven methodologies are already well-established in numerous fields such as astronomy, bio(techno)logy, drug discovery, quantitative social sciences, physics, and chemistry, the application of informatics in the domain of materials science, also termed “materials informatics”, is still in its infancy [9, 32, 34–39]. Materials informatics (MI), or the large-scale algorithmic analysis of materials data to gain novel insight, is a rapidly-growing field in materials science, and represents a groundbreaking way to integrate scientific knowledge, theory, and advanced computation for the discovery of new materials [9, 37, 38, 40, 41]. When combined with state-of-the-art machine learning (ML) algorithms, MI can be used to predict the properties of materials which do not currently exist, or be used to guide materials selection and design given a set of target material properties.

As machine learning frameworks and algorithms are developed, important advancements are not only being made by those doing research in these areas, but also by those who apply ML modeling methodologies within the materials science domain [34, 36, 40, 42, 43]. The Materials Genome Initiative (MGI), launched in 2011, is one example of where data science and ML methods are being applied to domain-specific knowledge in materials science. The MGI facilitates collaborations between experimental and computational materials scientists, with the goal of discovering, developing, manufacturing, and deploying new materials “twice as fast, at a fraction of the cost” [39, 44].

2.3 Machine learning for materials science

Machine learning (ML) is a subarea of AI with the aim of developing and using data-driven algorithms. In doing so, knowledge from data is automatically learned by training the model on it [45–47]. Although ML is often considered as a research area of conventional computer science, it differs from traditional computational approaches. In conventional computing, algorithms (explicitly programmed instructions) are used by computers to compute and solve specific and precisely-defined problems. ML and deep learning (DL)¹ algorithms, on the other hand, enable computers to train on data and develop the ideal model using methods from statistical analysis and data science. The trained model and the learned presentation of the data relationships can then be used to make predictions on previously unseen input data of the same type. In this regard, the rulesets for processing and representing input data in the trained models are not explicitly programmed, but rather learned on the basis of the data.

The integration of ML in materials science research workflows brings many benefits. ML models are able to recognize complex and non-linear relationships from large amounts of data, and are thus particularly applicable if the data or data relationships are too complex to be described or solved with conventional methods (whether analytically or computationally). In addition, the accuracy of a ML model improves when a large amount of representative data for model training is available. Another significant benefit of using ML over other first-principles computational materials science methods such as density functional theory (DFT) and molecular dynamics (MD) is the drastic speedup that machine learning can bring. Compared to DFT methods, which may take days or weeks to compute the property of a single material, ML methods can compute the properties of tens of thousands of materials, within seconds to minutes [48–50].

Thus, in the search for novel materials, high-throughput ML models can be used to screen for promising material compositions from a large number of candidate compositions (Figure 2.4). Using ML, researchers can better navigate the chemical whitespace to identify interesting candidate materials or additional directions for further research.

¹Deep learning is a subset of machine learning and is based on artificial neural networks inspired by the structure and function of the brain. In DL, multiple processing layers are used to transform the data into different levels of abstraction (representations) that extract progressively higher-level information. **Note:** since DL is a subset of ML, both ML and DL models/algorithms are meant when ML is mentioned in the text.

2.3 MACHINE LEARNING FOR MATERIALS SCIENCE

For example, from a list of 10^9 candidates, ML models can be used to identify 10^3 promising compositions. Subsequently, the results of the ML screening can be combined with computational or simulation methods to further narrow the number of candidate compositions down to 10^2 – 10^1 compositions. At the end of this ML-assisted, automated workflow, the large pool of candidates has already been narrowed down by more than 7 magnitudes without the need of any manual experimentation or expert intervention. The remaining 10^2 to 10^1 candidates can then be tested experimentally in the laboratory, drastically reducing time and research costs compared to simple trial and error experimentation [19, 26, 32].

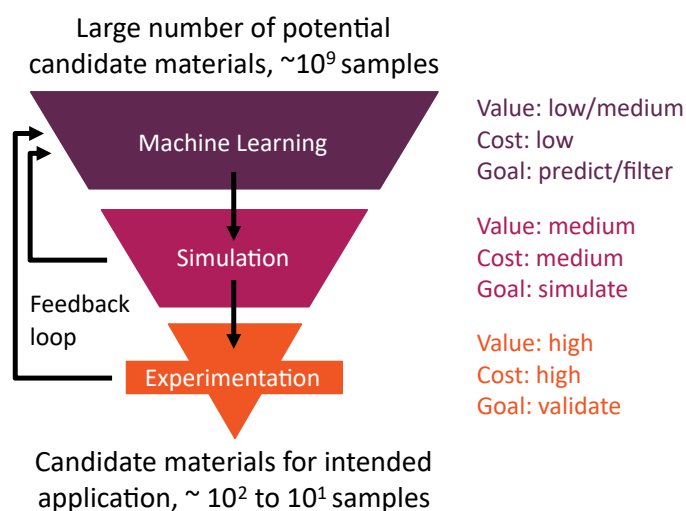


Figure 2.4: A typical workflow for materials discovery, integrating materials informatics, machine learning, simulation and experimentation to narrow down a large number of potential candidate materials to a few promising candidates. Figure adapted with permission from [51].

ML in MI has received significant attention in the academic research world and is gaining widespread adoption in the study of many inorganic material properties such as mechanical, electronic, thermodynamic, and transport properties. More specifically, it has recently been studied for its use in the research and design of novel inorganic materials in many different application areas such as photovoltaic materials [52–54], materials for energy storage [55–59], catalysts/photocatalysts [60–66], thermoelectric materials [21, 67–70], high-temperature superconductors [71–77], and high entropy [78, 79] and metallic glass alloys [52, 80, 81].

2 HUMANITY'S MATERIAL CHALLENGES

These studies were enabled by the increasing abundance of experimental and simulated materials property data available online through several materials data repositories as well as the improvements in the materials informatics ecosystem (such as algorithm and model developments, open source software, model hosting and sharing platforms). An overview of other impactful and relevant works in MI as well as the materials data repositories, software tools, and methodologies often used by researchers in the field is given as part of the first publication included in this dissertation (Chapter 3.2). Furthermore, the interested reader is encouraged to refer to the cited works as well as other well-written reviews available in the literature regarding this exciting research field [26, 27, 34, 36, 37, 40–42, 82–96].

The accelerating research interest in the field of MI is encouraging. However, many open questions and issues remain to be addressed in the adaptation of data-driven techniques for materials science. These are discussed in detail in the upcoming chapters.

3 Best practices for machine learning in materials science

3.1 The need for best practices in materials informatics

Despite the remarkable amount of interest in data-driven materials science and MI in the past ~15 years, they remain relatively new research fields with many unanswered questions [97]. The use and implementation of data science and ML methodologies are unfamiliar skills that are not typically required for the materials scientist. On the other hand, the proliferation and democratization of software packages make it appear easy to implement basic proof of concept studies using data science and ML in materials science. However, the inexperience with these methodologies in the materials science field often leads to published studies which, while interesting and potentially useful, are often misguided or incorrect¹. Without a firm understanding and careful consideration of the data processing, modeling, and model validation choices in these studies, the interested materials scientist may be misled into believing inaccurate results or adapting improper methodologies for follow-up works.

Oftentimes, it is apparent from the published works that inadequate attention is being paid to the fundamental assumptions and limitations of the methodologies used for data processing, data splitting, featurization, modeling, and evaluation. The model results are often blindly trusted and reported without a critical examination of their truthfulness. In other cases, modeling choices are not clearly explained—models are sometimes taken from other domains in ML and poorly adapted to the domain of materials science. While it is encouraging to see researchers getting inspiration and adapting modeling methods from other domains to material science, in many occurrences, the models appear to be more “transplanted into”, rather than being “adapted for” materials science.

¹“Garbage in, garbage out”, as a computer scientist would say.

Another major roadblock is in the lack of reproducibility in the works. In many early publications—and to an extent, still in some new publications today—the algorithms and methodologies used in the studies are not fully published or open-sourced. For some works, the datasets used are also not provided. Reproducibility in reporting is a major principle in good scientific practice. Especially for studies involving computational methods such as MI, it is simpler (requiring typically less specialized equipment or materials) to demonstrate and enable the reproducibility of results.

Furthermore, in many early works where modeling techniques were introduced, the researchers did not rely on standardized materials property datasets to evaluate the model performance. Instead, the researchers used in-house or private datasets to benchmark their models against baseline models. This use of a non-standardized dataset prevents the direct comparison of model results between different publications and produces additional hurdles for future researchers who wish to compare against existing models. Lastly, the use of different model evaluation metrics to present the model performance also prevents the quick comparison of model results between different publications.

Certainly, most of these cases are unintentional or are the result of incomplete understanding of the methodologies used, and are not a deliberate act of omission. Nevertheless, it is important for the field of MI that a basic set of guidelines and best practices is established with the goals of unifying research reporting, improving publication quality, and facilitating collaboration in this field. This will not only lend credibility to future publications, but also promote further research and development into this exciting field.

The first publication resulting from this dissertation work addresses these issues. The publication and its accompanying supplementary information (SI) are inserted in the following pages.

3.2 Publication 1: Machine Learning for Materials Scientists: An Introductory Guide toward Best Practices

Title	Machine Learning for Materials Scientists: An Introductory Guide toward Best Practices
Authors	A. Y.-T. Wang, R. J. Murdock, S. K. Kauwe, A. O. Oliynyk, A. Gurlo, J. Brgoch, K. A. Persson, and T. D. Sparks
Journal	Chemistry of Materials
Publisher	American Chemical Society (ACS) Publications
Publication date	May 19, 2020
Reference	<i>Chemistry of Materials</i> , 2020 , 32 (12): 4954–4965
DOI	10.1021/acs.chemmater.0c01907
Supporting information	The SI can be downloaded from: https://doi.org/10.1021/acs.chemmater.0c01907
My contribution	Conceptualization (with R.J.M. and S.K.K.), Methodology, Software, Validation, Investigation, Data Curation, Visualization, Writing – Original Draft, Writing – Review & Editing, Project administration.

The article is inserted in the following pages. Reprinted (adapted) with permission from (*Chemistry of Materials*, **2020**, 32 (12): 4954–4965). Copyright (2020) American Chemical Society. The article can also be downloaded using the ACS Articles on Request author-directed link: <https://pubs.acs.org/articlesonrequest/AOR-HCRBYMBIJ9NMYTVKDUXX>.



pubs.acs.org/cm

Methods/Protocols

Machine Learning for Materials Scientists: An Introductory Guide toward Best Practices

Anthony Yu-Tung Wang, Ryan J. Murdock, Steven K. Kauwe, Anton O. Oliynyk, Aleksander Gurlo, Jakoah Brgoch, Kristin A. Persson, and Taylor D. Sparks*



Cite This: *Chem. Mater.* 2020, 32, 4954–4965



Read Online

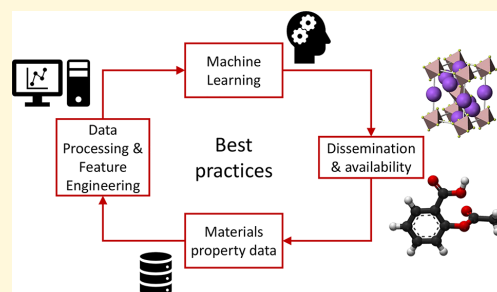
ACCESS |

Metrics & More

Article Recommendations

Supporting Information

ABSTRACT: This Methods/Protocols article is intended for materials scientists interested in performing machine learning-centered research. We cover broad guidelines and best practices regarding the obtaining and treatment of data, feature engineering, model training, validation, evaluation and comparison, popular repositories for materials data and benchmarking data sets, model and architecture sharing, and finally publication. In addition, we include interactive Jupyter notebooks with example Python code to demonstrate some of the concepts, workflows, and best practices discussed. Overall, the data-driven methods and machine learning workflows and considerations are presented in a simple way, allowing interested readers to more intelligently guide their machine learning research using the suggested references, best practices, and their own materials domain expertise.



INTRODUCTION

Materials scientists are constantly striving to advance their ability to understand, predict, and improve materials properties. Due to the high cost of traditional trial-and-error methods in materials research (often in the form of repeated rounds of material synthesis and characterization), material scientists have increasingly relied on simulation and modeling methods to understand and predict materials properties *a priori*. Materials informatics (MI) is a resulting branch of materials science that utilizes high-throughput computation to analyze large databases of materials properties to gain unique insights. More recently, data-driven methods such as machine learning (ML) have been adopted in MI to study the wealth of existing experimental and computational data in materials science, leading to a paradigm shift in the way materials science research is conducted.

However, there exist many challenges and “gotchas” when implementing ML techniques in materials science. Furthermore, many experimental materials scientists lack the know-how to get started in data-driven research, and there is a lack of recommended best practices for implementing such methods in materials science. As such, this article is designed to assist those materials science scholars who wish to perform data-driven materials research. We demonstrate a typical ML project step-by-step (Figure 1), starting with loading and processing data, splitting data, feature engineering, fitting different ML models, evaluating model performance, comparing performance across models, and visualizing the results. We

also cover sharing and publication of the model and architecture, with the goal of unifying research reporting and facilitating collaboration this emerging field. Throughout this process, we highlight some of the challenges and common mistakes encountered during a typical ML study in materials science, as well as approaches to overcome or address them. Highlighting the best practices will improve the research and manuscript quality and ensure reproducible results.

To demonstrate some of the best practices discussed throughout this work, we have created several interactive Jupyter notebooks with relevant Python code structured in a tutorial format (Table 1). The sections in this article that include accompanying notebooks are marked with an asterisk*. The notebooks walk the readers through a basic ML study in materials science: the prediction of heat capacity for solid inorganic compounds. We demonstrate this by implementing several classical machine learning as well as neural network models from the well-known Python packages scikit-learn and PyTorch, respectively. The Jupyter notebooks can be accessed at the online GitHub repository: <https://github.com/anthony-wang/BestPractices>. Setup, usage, further instructions, and

Received: May 5, 2020
Revised: May 19, 2020
Published: May 19, 2020



ACS Publications

© 2020 American Chemical Society

4954

<https://dx.doi.org/10.1021/acs.chemmater.0c01907>
Chem. Mater. 2020, 32, 4954–4965

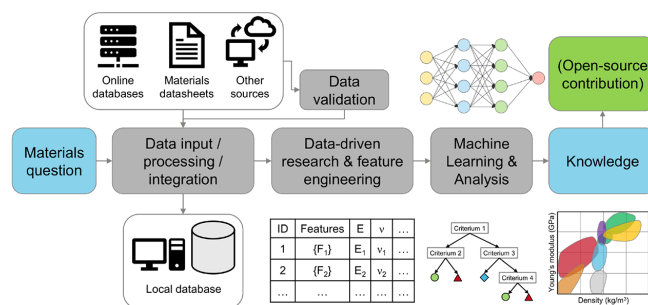


Figure 1. Schematic of a machine learning study in materials science.

Table 1. List of Accompanying Jupyter Notebooks and the Topics Demonstrated

no.	notebook contents
1	Loading data; examining, processing, cleaning up of data
2	Splitting data into train/validation/test data sets
3	Featurizing data; modeling with classical models, evaluating models, effect of different train/validation/test splits
4	Modeling with neural networks, evaluating models, exporting models, avoiding overfitting
5	Visualizing results

pertinent information can also be found there. Please note, an intermediate knowledge of the Python programming language and general programming principles is required.

MEANINGFUL MACHINE LEARNING

Machine learning is a powerful tool, but not every materials science problem is a nail. It is important to delineate when to use ML and when it may be more appropriate to use other methods. Consider what value ML can add to your project and whether there are more suitable approaches. Machine learning is most useful when human learning is impossible, such as where the data and interactions within the data are too complex and intractable for human understanding and conceptualization. Contrarily, machine learning often fails to find meaningful relationships and representations from small amounts of data, when a human mind would otherwise likely succeed.

When developing ML tools and workflows, consider how (and with what ease) they can be used not only by yourself but by others in the research community. If another researcher wants to use your method, will they be able to do so, and will it be worth it for them? For example, if you include data from *ab initio* calculations such as density functional theory (DFT), or crystal structure as one of the input features of your ML model, would it not be simpler for other researchers to use DFT or other simulation methods themselves, instead of using your ML model?

Another limitation to consider when using ML as a tool is the model interpretability vs predictive power trade-off. If you are looking for physical or chemical insights into your materials, you are unlikely to find them when using powerful and complex models such as neural networks: these models—while they can exhibit high model performance—are usually too complex to be easily understood. These are so-called “black-box” models because outside of their inputs and outputs, it is nearly impossible for a human to grasp the

inner model workings and its decision making processes. In contrast, simpler models might be easier to understand but tend to lack the predictive power of the more complex models.

In general, a good ML project should do one or more of the following: screen or down-select candidate materials from a pool of known compounds for a given application or property,^{1–3} acquire and process data to gain new insights,^{4,5} conceptualize new modeling approaches,^{6–10} or explore ML in materials-specific applications.^{1,11–13} Consider these points when you judge the applicability of ML for your project.

MACHINE LEARNING IN MATERIALS SCIENCE

Machine learning has been applied in the study of many inorganic material properties, such as mechanical, electronic, thermodynamic, and transport properties. It has also been used in many different material application areas, such as photovoltaic materials, materials for energy storage, catalysts/photocatalysts, thermoelectric materials, high-temperature superconductors, and high entropy and metallic glass alloys. We highlight some current examples in the literature of inorganic material properties and their application areas in Table 2. Here, we are not attempting to summarize the methods or results of these studies; instead, we advise the

Table 2. Examples of Using Machine Learning in the Study of Inorganic Materials

material properties	refs
Mechanical properties	1, 6, 9, 25–30
Formation energy	7, 9, 29, 31–34
Band gap	6, 9, 29, 35–39
Density of states	40, 41
Crystal structure/stability	32, 42–52
Debye temperature/heat capacity	6, 53, 54
Thermal expansion coefficient	6, 53
Thermal conductivity	6, 53, 55–57
Seebeck coefficient	56, 58
material classes	refs
Photovoltaic materials	34, 59, 60
Energy storage	61–65
Catalysts/photocatalysts	2, 66–71
Thermoelectric materials	4, 13, 56, 72, 73
High-temperature superconductors	74, 74–80
High entropy alloys	81, 82
Metallic glass alloys/glass-forming ability	34, 83, 84

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

Table 3. Comparison of Materials Data Repositories with Predominantly Property Information

name	structure information	mechanical properties	thermal properties	electronic properties	API ^a	data license	refs
Materials Project	Y	Y	Y	Y	Y	CC BY 4.0	85
Open Quantum Materials Database	Y	N	Y	Y	Y	CC BY 4.0	86
AFLOW for Materials Discovery	Y	Y	Y	Y	Y	<i>b</i>	87
Novel Materials Discovery (NOMAD)	Y	Y	Y	Y	Y	CC BY 4.0	88
Open Materials Database	Y	N	Y	Y	Y	CC BY 4.0	89
Citrine Informatics	Y	Y	Y	Y	Y	CC BY	90
Materials Platform for Data Science (MPDS)	Y	Y	Y	Y	Y	CC BY 4.0	91
AiiDA/Materials Cloud	Y	Y	Y	Y	Y	Varies	92, 93
NREL MatDB	Y	N	Y	Y	N	Own license	94
NIST TRC Alloy Data	N	N	Y	N	On request	Free	95
NIST TRC ThermoData	N	N	Y	N	N	NIST SRD	96
NIST JARVIS-DFT/-ML Database	Y	Y	Y	Y	Y	Public domain	97, 98
MatWeb	N	Y	Y	N	N	Paid	99
Total Materia	N	Y	Y	N	N	Paid	100
Anslys Granta (MaterialUniverse repository)	N	Y	Y	N	N	Paid	101
MATDAT	N	Y	Y	N	N	Paid	102

^aAn “application programming interface” is a set of defined functions, procedures, methods, or classes which enable a structured way of exchanging data between programs. In the framework of a materials data repository, an API facilitates, e.g., the uploading, examining, and downloading of data and other forms of interactions between the user and the repository. ^bNot specified.

interested reader to refer to the cited works as well as other well-written reviews available in the literature.^{3,14–24}

WORKING WITH MATERIALS DATA

Data Source. Some of the more commonly used repositories for materials property data are shown above in Table 3.

Other repositories that host predominantly crystal structure information are shown below in Table 4. While these repositories do not necessarily host material property information, the structure information contained within these repositories are also valuable.

Table 4. Comparison of Materials Data Repositories with Predominantly Structure Information

name	no. records ^a	API	Data license	ref
Cambridge Structural Database (CSD)	1,055,780	Y	Paid	103
Inorganic Crystal Structure Database (ICSD)	216,302	N	Paid	104
Pearson's Crystal Data (PCD)	335,000	N	Paid	105
International Centre for Diffraction Data (ICDD)	1,004,568	N	Paid	106
Crystallography Open Database (COD)	455,714	Y	Open-access	107
Pauling File	357,612	Y	Paid	108
CrystMet database	160,000	N	Paid	109

^aNote: values for number of records were updated as of the submission date (May 2020).

There is an ever-increasing number of materials informatics-related resources and repositories; as such, only the more commonly used repositories are mentioned above. Keep in mind that each data set is different and may contain domain-specific information and features that are restricted to certain research fields or applications. There may also be differences in the methodologies with which the data are experimentally or computationally derived, curated, and recorded in these databases. As a result of this, the values for material properties

might not be directly comparable across the different repositories. Be mindful of this when you are comparing property data across the repositories, especially if you plan on aggregating or merging data from different sources.

Data Set Size and Composition*. When collecting your data set for your ML study, be mindful of your data set size. Ensure that your data set size is large enough and includes most examples of the combinations of material compositions in the material space you want to study. It is also important to consider data balance or bias in your data sets. Does your data form clusters based on chemical formula, test condition, structure type, or other criteria? Are some clusters greatly over- or under-represented? Many statistical models used in ML are frequentist in nature and will be influenced by data set imbalance or bias. Visualization techniques such as t-distributed stochastic neighbor embedding (t-SNE¹¹⁰), uniform manifold approximation and projection (UMAP¹¹¹), or even simple elemental prevalence mapping¹¹² may be useful in investigating data set imbalance and bias.

Lastly, if your data set is too large (a rare luxury in the materials science field), you may find yourself having to wait a long time to train and validate your models during the prototyping phase of your project. In this case, you can subsample your data set into a small-scale “toy data set” and use that to test and adjust your models. Once you have tuned your models to your satisfaction on the toy data set, you can then carry on and apply them to the full data set. When sampling the original data set to create the toy data set, be aware that you do not introduce any data set biases through your sampling. Also keep in mind that not all performance-related problems can be fixed by subsampling your data. If your model can only train successfully on the toy data set and cannot train on the full data set (e.g., due to memory or time constraints), you may wish to focus on improving its performance first.

Data Version Control. Be sure to save an archival copy of your raw data set as obtained and be sure that you can retrieve it at any time. If you make any changes to your data set, clearly record the steps of the changes and ensure that you are able to

3.2 PUBLICATION 1: BESTPRACTICES

reproduce them on the data set in the future if needed. To simplify version control, consider using a version control system (such as Git,¹¹³ Mercurial,¹¹⁴ or Subversion¹¹⁵) for your data sets.

Cleanup and Processing*. Once you have curated your data set, examine and explore the data on a high level to see if there are any obvious flaws or issues. These may—and often do—include missing or unrealistic values (e.g., NaN's, or negative values/positive values where you do not expect them), outliers or infinite values, badly formatted or corrupt values (e.g., wrong text encoding, numbers stored in non-numeric format), nonmatching data formats or data schema caused by changes in the repository, and other irregularities. If you find any irregularities, deal with them in an appropriate way and be careful not to introduce any bias or irregularities of your own. Make sure you document any data cleanup and processing steps you performed; this is an important step in ensuring reproducibility that is often overlooked in ML studies. In addition, during your model prototyping stage, you may find some additional problematic data samples which adversarially affect your model performance. In this case, consider performing another round of data cleanup before finalizing your model.

Train–Validation–Test Split*. Split your data once into three data sets: train, validation, and test. The split should be performed in a reproducible way (e.g., by assigning a random seed and shuffling the data set); alternatively, you can save the split data sets as files for reuse. Make sure that no same (or similar) data appear in the test data set, if they are already present in the train or validation data set. For example, if you have several measurements of a chemical compound that are performed at different measurement conditions in the train data set (e.g., temperature or pressure), during the testing phase, your model would likely perform well if it is asked to predict the property of the same compound at a different condition. This, however, gives you an inflated estimate of how well the model will generalize in cases where it has not seen a particular chemical compound before. For a truly rigorous evaluation of your model's generalization performance, you should take care to avoid this data leakage when you split your data.

During the training stage, models may only be shown the training data as part of the learning process. Validation data may be used to assess and tune different model hyperparameters and may be compared with the predictions of different model/hyperparameter combinations to evaluate a model's performance. In contrast, test data may only be used in order to evaluate a model's performance as a final step, after the model has been finalized. Models *must not* be trained *nor* tuned on the test data set. Use the same train, validation, and test data sets for all modeling and model comparison/benchmarking steps.

The training data set can be further partitioned to be used for cross-validation (CV). CV is a method that is often employed to estimate the true ability of a model to predict on new unseen data and to catch model-specific problems such as overfitting or selection bias.¹¹⁶ One typical method is *k*-fold cross-validation. In *k*-fold CV, the training data set is first randomly partitioned into *K* subsets (remember to note down your partitioning details). Then, for each *k* of the data subsets *k* = 1, 2, ..., *K*, the model is trained on the combined data of the other *K* – 1 subsets and then evaluated using the *k*th subset. The resulting *K* prediction errors are then typically averaged to

give a more accurate estimate of the model's true predictive performance compared to evaluating the model performance on one single train/validation/test split. Typical choices for *K* in the literature are 5 or 10. In the case of a small input data set size, *k*-fold CV or other methods of cross-validation can also be used as a data resampling technique for models that are more robust against overfitting on the validation set (e.g., linear regression).

■ MODELING

Choosing Appropriate Models and Features*. The data set size will almost always determine your available choices of ML models. For smaller data set sizes, classical and statistical ML approaches (e.g., regression, support vector machines, *k*-nearest neighbors, and decision trees) are more suitable. In contrast, neural networks require larger amounts of data and only start becoming feasible/useful when you have training data points on the order of thousands or more. Typically, ML models such as regression, decision tree/random forest, *k*-nearest neighbors, and support vector machines are used on smaller data sets. These algorithms can be further improved by applying bagging, boosting, or stacking approaches. There are many existing Python libraries for implementing the above, with perhaps the most well-known being scikit-learn.¹¹⁷ For larger data sets, neural networks and deep learning methods are more commonly used. In the scholarly community, the Python libraries PyTorch¹¹⁸ and TensorFlow¹¹⁹ are often used to implement these architectures.

Feature engineering is important for smaller data set sizes and can contribute to a large model performance increase if the features are well-engineered.^{1,54,120} A common way to transform chemical compositions into usable input features for ML studies is through the use of composition-based feature vectors ("CBFVs"). There are numerous forms of the CBFV available, such as Jarvis,¹²¹ Magpie,³⁴ mat2vec,⁴ and Oliynyk.¹³ These CBFVs contain values that are either experimentally derived, calculated through high-throughput computation, or extracted from materials science literature using ML techniques. Instead of featurizing your data using CBFVs, you can also try a simple onehot-encoding of the elements. These CBFV featurization schemes as well as the relevant functions and code for featurizing chemical compositions are included in the online GitHub repository associated with this work.

For sufficiently large data sets and for more "capable" learning architectures like very deep, fully connected networks^{7,122} or novel attention-based architectures such as CrabNet,⁶ feature engineering and the integration of domain knowledge (such as through the use of CBFVs) in the input data becomes irrelevant and does not contribute to a better model performance compared to a simple onehot-encoding.¹¹ Therefore, due to the effort required to curate and evaluate domain knowledge-informed features specific to your research, you may find it more beneficial to seek out additional sources of data and already-established featurization schemes or use learning methods that do not require domain-derived features⁶ instead.

Data Scaling and Normalization*. In most cases, it may be beneficial to scale your input data (*X*). For a regression task, it may also be helpful to scale the targets (*y*) as well. Scaling can be done in many ways. Often, the input data is scaled to have zero-mean and unit variance. This allows for more stable

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

gradients and faster model convergence, since the resulting feature dimensions are similar in scale.^{123–126}

This is done by using the transformation:

$$X' = (X - \bar{X})/\sigma_X \quad (1)$$

where \bar{X} denotes the mean and σ_X the standard deviation of X . In some cases, applying the logarithm function to your values before scaling them according to eq 1 may further improve your model performance.

Keep in mind that the scaling operations must be conducted using solely the statistics from the training data set (i.e., the train/validation/test data sets are scaled using only the mean and standard deviation values computed from the training data) and that the validation and test data statistics must not be used. Remember also to undo the scaling operation(s) on the target values (if these were scaled) after loss computation but before performance evaluation. Similar to scaling, normalization of X is recommended for regression tasks. Here it is also important to use only the training data statistics when normalizing input data.

Scaling and normalization are not commutable: their ordering matters. You should scale, then normalize. When undoing this operation, the inverse is required: unnormalize and then unscale.

Keep It Simple. Sometimes, especially in the case of small data set sizes, simpler models can perform better than more complex models on the held-out test data. Some simpler models that you can try are linear (or ridge/lasso) regression, random forest, or k -nearest neighbors.

Furthermore, consider the model complexity–explainability trade-off. Typically, more complex models achieve higher model performance but have the caveat that they are generally not easily interpretable by humans. In contrast, simpler models are typically assumed to be more easily understood by humans and lead to better opportunities for model introspection. This is an important consideration in materials science, since synthesis and characterization are costly and time-consuming and the costs must be justified.

Hyperparameter Optimization. Depending on your choice of ML model, there may be model hyperparameters that can be tuned. Examples of hyperparameters are the number of neighbors (k) in k -nearest neighbors, the number and depth of trees in a random forest, the kernel type and coefficient in support vector machines, the maximum number of features to consider in gradient boosting, and loss criterion, learning rate, and optimizer type in neural networks. These hyperparameters are properties of the models themselves and can significantly affect your model's performance, speed (in training and inference), and complexity.

The hyperparameters are not learned by the model during the training step; rather, they are selected by you when you create the model. The recommended way to optimize your model hyperparameters is by training numerous models (each with a different set of hyperparameters) using the same training set and then evaluating the models' performance using the same validation set. By doing this, you will be able to identify the set of hyperparameters that generally leads to good-performing models. This is commonly referred to as a "grid search". Imagine that your model has two continuous-variable hyperparameters, h_1 and h_2 , and that there is a range of values for each of these parameters that you wish to investigate, $[h_{1,\min}, h_{1,\max}]$ and $[h_{2,\min}, h_{2,\max}]$, respectively. You can then define a grid that spans between $(h_{1,\min}, h_{2,\min})$ and $(h_{1,\max},$

$h_{2,\max})$. At each point on this grid, you train a model corresponding to that set of hyperparameters using the training set and then evaluate its performance on the validation set. After repeating this for every point on the grid, you obtain a mapping that you can then use to determine the best set of hyperparameters for your specific model and data.

Once again, we stress the importance of reserving a held-out test data set during data set splitting. By training and optimizing your model on the training and validation data sets, you have effectively tuned—and possibly biased—your model to perform exceptionally well on these data samples. Therefore, the performance metrics of your model on these data sets are no longer good indicators of your model's true generalization ability. In contrast, evaluating your model's performance on the held-out test data set (which your model has never seen before) will give you a much more realistic estimate.

Model Evaluation and Comparison. Typically, studies in materials science will compare the performance of several ML model and hyperparameter combinations on a given task. Trained models are typically compared by evaluating their performance on the held-out test data set using computed test metrics such as accuracy, logarithmic loss, precision, recall, F1-score, ROC (receiver operating characteristic curve), and AUC (area under curve) for classification tasks and r^2 (Pearson correlation coefficient), mean absolute error, and (root) mean squared error for regression tasks. Also consider using cross-validation (as discussed earlier) to give a more accurate estimate of your model's true performance.

Show Your Model*. If you are reporting a new model architecture or algorithm, you must include all pertinent information necessary to reproduce, evaluate, and apply your models. This entails providing the complete source code for your implementation, the hyperparameters used, the random seeds applied (if any), and the pretrained weights of the models themselves. In addition, clear descriptions and schematics of your new system should be provided, as well as instructions to reproduce your model and work. Ideally, you can show your model and results in an interactive manner, such as through the use of Jupyter notebooks.

■ FITTING AND TESTING

Avoid Overfitting*. In an ML problem, the model is asked to perform two contradicting tasks: (1) minimize its prediction error on the training data set and (2) maximize its ability to generalize on unseen data. Depending on how the model, loss criterion, and evaluation methods are set up, the model may end up memorizing the training data set (an unwanted outcome) rather than learning an adequate representation of the data (the intended outcome). This is called "overfitting" and usually leads to decreased generalization performance of the model. Overfitting can occur on all kinds of models, although it typically occurs more often on complex models such as random forests, support vector machines, and neural networks.

During model training, observe the training metrics such as your loss output and r^2 score on the training and validation set. For example, when training a neural network, you can use a learning curve to track validation errors over each epoch during the training process. As the model trains, the validation and training error will ideally decrease. Your training error will approach zero, but this is not the metric we care about! Rather, you should closely observe the validation error. When your

validation error increases again while your training error continues to decrease, you are likely memorizing your training data and thus overfitting your data.

Overfitting can have an adverse effect on your model's ability to generalize (that is, returning a reasonable output prediction for new and unseen data), thus performing poorer on the test data set. If you notice that your model overfits your data very easily, consider reducing the complexity of your model or using regularization.

Beware of Random Initialization*. Many ML models require an initial guess as a starting point for their internal parameters. In many model implementations (e.g., in scikit-learn's linear regression, random forest, support vector machines, boosting implementations), these initial internal model parameters are provided by your system's random number generator. The same applies for neural network-based models, in the initialization of the weights and biases of the networks and some optimizer parameters. Depending on how sensitive your model is to initialization, different initial states of the models can lead to significant differences in your model performance.

It is therefore important that you ensure reproducible results across different model runs and different models (both for your internal testing and for publication). To accomplish this, you can choose a seed to use for the random number generator. Do not forget to mention this seed in your publication and code. Note that alternative ways of model initialization exist, such as using different estimators for initial parameter guesses as well as different initialization schemes for neural network weights and biases; here, you should note down your changes if you use an alternative implementation.

Avoid *p*-Hacking. Train your models on the training data set only and use the validation data set for tuning your model hyperparameters. Do not evaluate your model on the held-out test data set until you have finished tuning your model and it is ready for publication. Looking at the test data set multiple times to pick ideal model hyperparameters is a form of *p*-hacking and is considered cheating!¹²⁷

BENCHMARKING AND TESTING

Reproducibly Testing Various Methods*. For comparison/ablation studies against other ML models and/or architectures, make sure you use the same train/validation/test data sets (refer to above for best practices on data set splitting and management). For the most informative and fair comparison between different published models, consider running the models yourself. If you perform any additional model-specific data manipulation steps, make sure to document them and make them reproducible for your readers.

During the model tuning process, train your models on the train data set and evaluate their performances on the validation set. After you have finalized your model architecture and hyperparameters, train the models once more on the combined train and validation data sets and evaluate their performances on the test data set.

Existing Benchmarks. There are some tools and software packages online that can be used as baselines to judge the performance of your models.^{128–131} Some of these tools can perform automatic feature engineering and testing of several different ML models. We suggest that you download these tools and compare the performance of your models against them. If your model does not perform better or does not offer

any advantages over these existing tools, consider other venues of improvement.

MAKING PUBLICATION-READY, REPRODUCIBLE WORK

Source Code and Documentation*. Publishing in peer-reviewed journals relies on the foundational principle that the

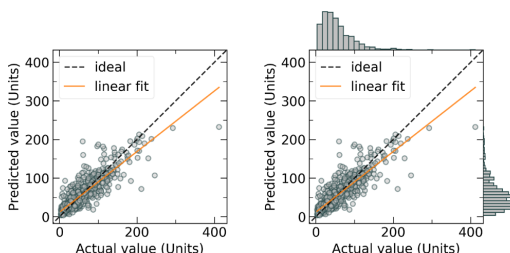


Figure 2. Example predicted vs actual material property plots, plotted (left) without and (right) with a marginal histogram. In addition, lines corresponding to ideal predictions (where the predicted values exactly match the actual values) and a linear regression fit (for estimating the correlation between the predicted and actual values) are shown.

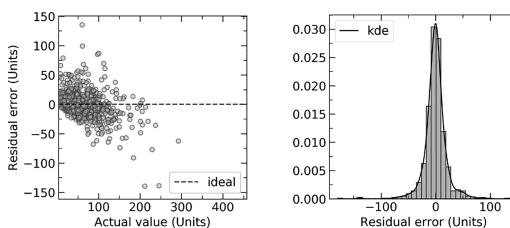


Figure 3. Example residual error plots, plotted (left) against the actual value and (right) as a histogram with a kernel density estimation (kde). A lower error indicates a more accurate model prediction.

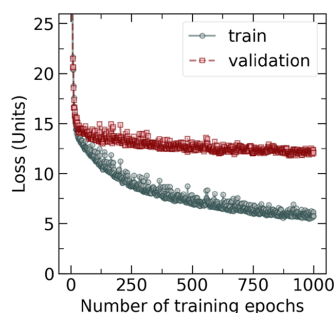


Figure 4. Example loss curve plot of a neural network, showing model performance (loss) evaluated on the train and validation data sets at each epoch throughout the training process. A lower loss indicates a better-performing model.

methodology be sufficiently described in order to ensure reproducibility. Therefore, for your ML-based study, full source code for your models and architecture (if any) must be provided, including implementation details of data processing, data cleanup, data splitting, model training, and

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

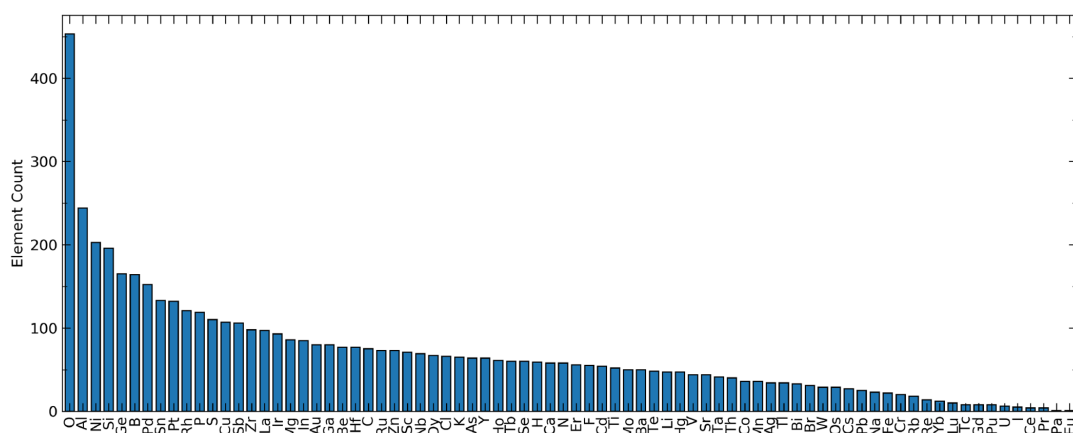


Figure 5. Example visualization of element prevalence in a data set, shown as a histogram.

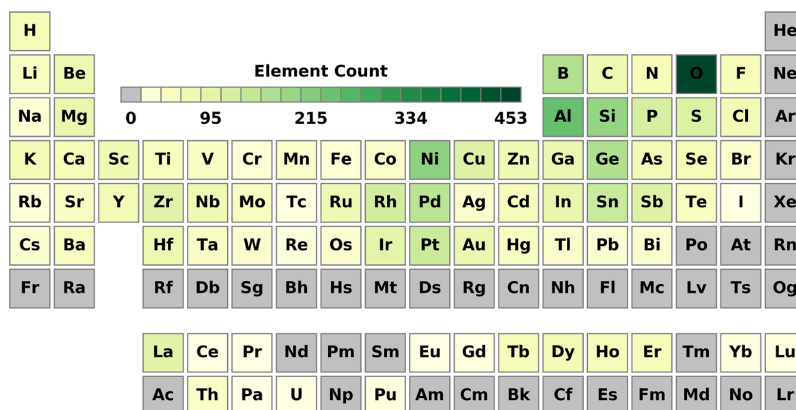


Figure 6. Example visualization of element prevalence in a data set, shown as a heatmap on a periodic table.

model evaluation. If you can, you should also publish your source code under a permissive or an open-source license so that others may (re)use, improve, collaborate on, and contribute further to your work.¹³²

Your published source code must be complete—that is, somebody should be able to take your source code verbatim, execute it, and obtain the same results that you did. Required libraries and other software dependencies (if any) must be listed, preferably with the pertinent version numbers. Ideally, these dependencies will be listed in an “environment file” that others can use to directly create a working software environment on their local system. If you use any code or packages developed by others, make sure to adhere to their licenses. Also consider hosting your code in an online, version-controlled repository such as GitHub, GitLab, Bitbucket, DHub,¹³³ or similar.

Make sure the source code is well-documented and follows well-established code standards. Instead of writing additional comments to explain your code, consider writing code in a way such that it is self-explanatory without the need for additional comments. This entails using clear variable names, closely following formatting guidelines (such as PEP 8), and writing

“explicit” code. Add a “README” file as well that provides your readers with instructions for the installation, setup, and usage of your code and for the reproduction of your published results. To ensure large-scale deployability and consistency on any infrastructure, consider also publishing your project as a containerized application, using tools such as Docker.¹³⁴

All Data Should Be Provided*. All results and data sets reported in the manuscript should be provided with the manuscript; alternatively, code for the users to obtain the data themselves must be given, ideally with clear instructions of the process. Additionally, all raw data—if their licenses allow it—should be provided with the manuscript as well. In the case where the data cannot be provided, due to licensing, legal and intellectual property protection, or other insurmountable hurdles, an explanation should be given. You are nevertheless encouraged to find alternative solutions for providing data within reason. Examples may be to provide a partial data set, an anonymized data set, trained model weights, or instructions for users on how to obtain the data set themselves. Consult with the owner of the data before considering these approaches, and as always, make sure you adhere to the data license.

3.2 PUBLICATION 1: BESTPRACTICES

Trained Models and Weights*. Ideally, you should provide a record of all model hyperparameters tested, as well as the best hyperparameters reported. For neural network implementations, the trained weights from the models should also be provided. In this case, be sure to provide the necessary code to recreate the neural network architecture and to load the saved weights for use. Ideally, you should also offer a friendly way to make predictions on user-supplied input data using these saved weights.

Visualizations*. All visualizations shown in the manuscript should be reproducible by a user who accesses your code. Ensure that you have included the required data (and ideally the code) used to generate the visualizations or have given the users a way to obtain the required data themselves. If there are additional figures, such as in the [Supporting Information \(SI\)](#), ensure that they are understandable by themselves and do not require additional explanation. If they do require explanation, provide this in the [SI](#) along with the figures.

Some of the typical visualizations that have shown themselves to be generally useful—and are thus commonly shown—in MI studies are predicted property value vs actual property value plots ([Figure 2](#)), residual error plots and histograms of residual errors ([Figure 3](#)), loss curves throughout the training process of a neural network ([Figure 4](#)), and element prevalence visualizations ([Figures 5 and 6](#)).

BENCHMARK DATA SETS

While there are currently several materials property data sets online which could potentially be used as benchmark data sets for benchmarking model performance in MI, there exist few published train/validation/test splits of these data sets which can be used by researchers to conduct a fair benchmark. Here, we note that examples of such data sets are commonly found in the fields of computer vision (e.g., CIFAR, Google Open Images data set, CelebFaces, ImageNet) as well as in natural language processing (e.g., Glue, decaNLP, WMT 2014 EN-DE).

Furthermore, the data heterogeneity—in terms of the classes of materials, the reported material properties, or the diversity in the types of compounds and constituent elements—of the available materials data sets are typically quite limited and vary from data set to data set. Additionally, the methods to access some of the data stored in the online data repositories are sometimes restricted and, therefore, hinder potential MI studies. This is due in part to the fact that certain data sets are proprietary or licensed under terms that do not allow their sharing (whether online or offline).

Another challenge is that the online material property repositories do not offer a “checkpointed” repository state; therefore, the repository and its data may change at any point in time, and there is no easy way to revert or refer back to the state of the repository at an earlier time. Therefore, current ML researchers typically download materials data sets from the repositories and archive them locally to run their benchmarks internally. However, there are recent emerging works from researchers that aim to address this issue of missing benchmark data sets for MI and ML studies in materials science.^{131,135,136}

SUMMARY

While various machine learning methods, including classical methods and more advanced techniques such as deep learning and neural network-based architectures, have successfully been

used for the prediction of materials properties, unique challenges still exist for their application in the domain of materials informatics. There are common pitfalls in the gathering, analysis, and reporting of materials science-related data and machine learning results and in the facilitation of reproduction studies. This Methods/Protocols article highlights a large number of these issues which are found in submitted manuscripts and published works in the field of materials informatics. Proper observation of the recommendations given above will certainly ensure higher publication standards and more reproducible science in this exciting emerging field.

ASSOCIATED CONTENT

Supporting Information

Online GitHub repository with the interactive Jupyter notebook files, Python source code, and example data are available at <https://github.com/anthony-wang/BestPractices>. The Supporting Information is available free of charge at <https://pubs.acs.org/doi/10.1021/acs.chemmater.0c01907>.

Read-only versions of the Jupyter notebook files ([PDF](#))

AUTHOR INFORMATION

Corresponding Author

Taylor D. Sparks — Department of Materials Science & Engineering, University of Utah, Salt Lake City, Utah 84112, United States; orcid.org/0000-0001-8020-7711; Phone: +1-801-581-8632; Email: sparks@eng.utah.edu

Authors

Anthony Yu-Tung Wang — Fachgebiet Keramische Werkstoffe/Chair of Advanced Ceramic Materials, Technische Universität Berlin, 10623 Berlin, Germany; orcid.org/0000-0002-7947-0309

Ryan J. Murdock — Department of Materials Science & Engineering, University of Utah, Salt Lake City, Utah 84112, United States

Steven K. Kauwe — Department of Materials Science & Engineering, University of Utah, Salt Lake City, Utah 84112, United States

Anton O. Oliynyk — Department of Chemistry & Biochemistry, Manhattan College, Riverdale, New York 10471, United States; orcid.org/0000-0003-0732-7340

Aleksander Gurlo — Fachgebiet Keramische Werkstoffe/Chair of Advanced Ceramic Materials, Technische Universität Berlin, 10623 Berlin, Germany; orcid.org/0000-0001-7047-666X

Jakoah Brgoch — Department of Chemistry, University of Houston, Houston, Texas 77204, United States; orcid.org/0000-0002-1406-1352

Kristin A. Persson — Energy Storage and Distributed Resources Division, Lawrence Berkeley National Laboratory, Berkeley, California 94720, United States; Department of Materials Science, University of California Berkeley, Berkeley, California 94720, United States; orcid.org/0000-0003-2495-5509

Complete contact information is available at:
<https://pubs.acs.org/10.1021/acs.chemmater.0c01907>

Notes

The authors declare no competing financial interest.

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

■ ACKNOWLEDGMENTS

A.Y.-T.W. and A.G. gratefully acknowledge support from the BIMO graduate school of the Technische Universität Berlin, the German Academic Exchange Service (Program No. 57438025), and the Deutsche Forschungsgemeinschaft. T.D.S. and S.K.K. are supported by the National Science Foundation (CMMI-1562226 and DMR-1651668) as well as the INL Laboratory Directed Research & Development (LDRD) Program under DOE Idaho Operations Office Contract DE-AC07-05ID145142. J.B. is supported by the National Science Foundation (DMR 18-47701 and CER 19-11311) as well as the Welch Foundation (E-1981). K.A.P. is supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, Materials Sciences and Engineering Division under Contract No. DE-AC02-05CH11231 (Materials Project program KC23MP). A.O.O. thanks Manhattan College for the support with start-up funds and Kakos Center for Scientific Computing at Manhattan College for providing computational resources.

■ REFERENCES

- (1) Mansouri Tehrani, A.; Oliynyk, A. O.; Parry, M.; Rizvi, Z.; Couper, S.; Lin, F.; Miyagi, L.; Sparks, T. D.; Brgoch, J. Machine Learning Directed Search for Ultrahard Materials. *J. Am. Chem. Soc.* **2018**, *140*, 9844–9853.
- (2) Singh, A. K.; Montoya, J. H.; Gregoire, J. M.; Persson, K. A. Robust and synthesizable photocatalysts for CO₂ reduction: a data-driven materials discovery. *Nat. Commun.* **2019**, *10*, 443.
- (3) Tabor, D. P.; Roch, L. M.; Saikin, S. K.; Kreisbeck, C.; Sheberla, D.; Montoya, J. H.; Dwaraknath, S. S.; Aykol, M.; Ortiz, C.; Tribukait, H.; Amador-Bedolla, C.; Brabec, C. J.; Maruyama, B.; Persson, K. A.; Aspuru-Guzik, A. Accelerating the discovery of materials for clean energy in the era of smart automation. *Nature Reviews Materials* **2018**, *3*, 5–20.
- (4) Tshitoyan, V.; Dagdelen, J.; Weston, L.; Dunn, A.; Rong, Z.; Kononova, O.; Persson, K. A.; Ceder, G.; Jain, A. Unsupervised word embeddings capture latent knowledge from materials science literature. *Nature* **2019**, *571*, 95–98.
- (5) Kim, E.; Huang, K.; Saunders, A.; McCallum, A.; Ceder, G.; Olivetti, E. Materials Synthesis Insights from Scientific Literature via Text Extraction and Machine Learning. *Chem. Mater.* **2017**, *29*, 9436–9444.
- (6) Wang, A. Y.-T.; Kauwe, S. K.; Murdock, R. J.; Sparks, T. D. Compositionally-Restricted Attention-Based Network for Materials Property Prediction: CrabNet. *ChemRxiv*, 2020. DOI: 10.26434/chemrxiv.11869026, accessed May 5, 2020.
- (7) Jha, D.; Ward, L.; Paul, A.; Liao, W.-K.; Choudhary, A.; Wolverton, C.; Agrawal, A. ElemNet: Deep Learning the Chemistry of Materials From Only Elemental Composition. *Sci. Rep.* **2018**, *8*, 17593.
- (8) Schütt, K. T.; Sauceda, H. E.; Kindermans, P.-J.; Tkatchenko, A.; Müller, K.-R. SchNet – A deep learning architecture for molecules and materials. *J. Chem. Phys.* **2018**, *148*, 241722.
- (9) Xie, T.; Grossman, J. C. Crystal Graph Convolutional Neural Networks for an Accurate and Interpretable Prediction of Material Properties. *Phys. Rev. Lett.* **2018**, *120*, 145301.
- (10) Goodall, R. E. A.; Lee, A. A. Predicting materials properties without crystal structure: Deep representation learning from stoichiometry. *arXiv*, 2019. <http://arxiv.org/pdf/1910.00617v2>, accessed May 5, 2020.
- (11) Murdock, R. J.; Kauwe, S. K.; Wang, A. Y.-T.; Sparks, T. D. Is Domain Knowledge Necessary for Machine Learning Materials Properties? *ChemRxiv*, 2020. DOI: 10.26434/chemrxiv.11879193, accessed May 5, 2020.
- (12) Kauwe, S. K.; Graser, J.; Murdock, R. J.; Sparks, T. D. Can machine learning find extraordinary materials? *Comput. Mater. Sci.* **2020**, *174*, 109498.
- (13) Oliynyk, A. O.; Antono, E.; Sparks, T. D.; Ghadbeigi, L.; Gaultois, M. W.; Meredig, B.; Mar, A. High-Throughput Machine-Learning-Driven Synthesis of Full-Heusler Compounds. *Chem. Mater.* **2016**, *28*, 7324–7331.
- (14) Lookman, T.; Alexander, F. J.; Rajan, K. *Information Science for Materials Discovery and Design*; Springer Series in Materials Science; Springer: Cham, Switzerland, 2016; Vol. 225.
- (15) Mueller, T.; Kusne, A. G.; Ramprasad, R. In *Reviews in Computational Chemistry*; Parrill, A. L., Lipkowitz, K. B., Eds.; Reviews in Computational Chemistry; John Wiley & Sons, Inc.: Hoboken, NJ, 2016; Vol. 1; pp 186–273.
- (16) Liu, Y.; Zhao, T.; Ju, W.; Shi, S. Materials discovery and design using machine learning. *Journal of Materiomics* **2017**, *3*, 159–177.
- (17) Gorai, P.; Stevanović, V.; Toberer, E. S. Computationally guided discovery of thermoelectric materials. *Nature Reviews Materials* **2017**, *2*, 17053.
- (18) Butler, K. T.; Davies, D. W.; Cartwright, H.; Isayev, O.; Walsh, A. Machine learning for molecular and materials science. *Nature* **2018**, *559*, 547–555.
- (19) Ramakrishna, S.; Zhang, T.-Y.; Lu, W.-C.; Qian, Q.; Low, J. S. C.; Yune, J. H. R.; Tan, D. Z. L.; Bressan, S.; Sanvito, S.; Kalidindi, S. R. Materials Informatics. *Journal of Intelligent Manufacturing* **2019**, *30*, 2307.
- (20) Rickman, J. M.; Lookman, T.; Kalinin, S. V. Materials informatics: From the atomic-level to the continuum. *Acta Mater.* **2019**, *168*, 473–510.
- (21) Gomes, C. P.; Selmán, B.; Gregoire, J. M. Artificial intelligence for materials discovery. *MRS Bull.* **2019**, *44*, 538–544.
- (22) Ong, S. P. Accelerating materials science with high-throughput computations and machine learning. *Comput. Mater. Sci.* **2019**, *161*, 143–150.
- (23) Schmidt, J.; Marques, M. R. G.; Botti, S.; Marques, M. A. L. Recent advances and applications of machine learning in solid-state materials science. *npj Computational Materials* **2019**, *5*, 83.
- (24) Meredig, B. Five High-Impact Research Areas in Machine Learning for Materials Science. *Chem. Mater.* **2019**, *31*, 9579–9581.
- (25) Bhadeshia, H. Computational design of advanced steels. *Scr. Mater.* **2014**, *70*, 12–17.
- (26) Agrawal, A.; Deshpande, P. D.; Cecen, A.; Basavarsu, G. P.; Choudhary, A. N.; Kalidindi, S. R. Exploration of data science techniques to predict fatigue strength of steel from composition and processing parameters. *Integrating Materials and Manufacturing Innovation* **2014**, *3*, 90–108.
- (27) Furmanchuk, A.; Agrawal, A.; Choudhary, A. Predictive analytics for crystalline materials: bulk modulus. *RSC Adv.* **2016**, *6*, 95246–95251.
- (28) de Jong, M.; Chen, W.; Notestine, R.; Persson, K. A.; Ceder, G.; Jain, A.; Asta, M.; Gamst, A. A Statistical Learning Framework for Materials Science: Application to Elastic Moduli of k-nary Inorganic Polycrystalline Compounds. *Sci. Rep.* **2016**, *6*, 34256.
- (29) Chen, C.; Ye, W.; Zuo, Y.; Zheng, C.; Ong, S. P. Graph Networks as a Universal Machine Learning Framework for Molecules and Crystals. *Chem. Mater.* **2019**, *31*, 3564–3572.
- (30) Evans, J. D.; Coudert, F.-X. Predicting the Mechanical Properties of Zeolite Frameworks by Machine Learning. *Chem. Mater.* **2017**, *29*, 7833–7839.
- (31) Meredig, B.; Agrawal, A.; Kirklín, S.; Saal, J. E.; Doak, J. W.; Thompson, A.; Zhang, K.; Choudhary, A.; Wolverton, C. Combinatorial screening for new materials in unconstrained composition space with machine learning. *Phys. Rev. B: Condens. Matter Mater. Phys.* **2014**, *89*, 094104.
- (32) Ghiringhelli, L. M.; Vybiral, J.; Levchenko, S. V.; Draxl, C.; Scheffler, M. Big data of materials science: critical role of the descriptor. *Phys. Rev. Lett.* **2015**, *114*, 105503.
- (33) Deml, A. M.; O'Hayre, R.; Wolverton, C.; Stevanović, V. Predicting density functional theory total energies and enthalpies of

3.2 PUBLICATION 1: BESTPRACTICES

formation of metal-nonmetal compounds by linear regression. *Phys. Rev. B: Condens. Matter Mater. Phys.* **2016**, 93, 085142.

(34) Ward, L.; Agrawal, A.; Choudhary, A.; Wolverton, C. A general-purpose machine learning framework for predicting properties of inorganic materials. *npj Computational Materials* **2016**, 2, 16028.

(35) Dey, P.; Bible, J.; Datta, S.; Broderick, S.; Jasinski, J.; Sunkara, M.; Menon, M.; Rajan, K. Informatics-aided bandgap engineering for solar materials. *Comput. Mater. Sci.* **2014**, 83, 185–195.

(36) Pilania, G.; Mannodi-Kanakthodi, A.; Uberuaga, B. P.; Ramprasad, R.; Gubernatis, J. E.; Lookman, T. Machine learning bandgaps of double perovskites. *Sci. Rep.* **2016**, 6, 19375.

(37) Sparks, T. D.; Kauwe, S. K.; Welker, T.; Sparks, T.; Kauwe, S. Extracting Knowledge from DFT: Experimental Band Gap Predictions Through Ensemble Learning. *ChemRxiv*, 2018. DOI: 10.26434/chemrxiv.7236029, accessed May 5, 2020.

(38) Rajan, A. C.; Mishra, A.; Satsangi, S.; Vaish, R.; Mizuseki, H.; Lee, K.-R.; Singh, A. K. Machine-Learning-Assisted Accurate Band Gap Predictions of Functionalized MXene. *Chem. Mater.* **2018**, 30, 4031–4038.

(39) Zhuo, Y.; Mansouri Tehrani, A.; Brgoch, J. Predicting the Band Gaps of Inorganic Solids by Machine Learning. *J. Phys. Chem. Lett.* **2018**, 9, 1668–1673.

(40) Schütt, K. T.; Glawe, H.; Brockherde, F.; Sanna, A.; Müller, K.-R.; Gross, E. K. U. How to represent crystal structures for machine learning: Towards fast prediction of electronic properties. *Phys. Rev. B: Condens. Matter Mater. Phys.* **2014**, 89, 205118.

(41) Yeo, B. C.; Kim, D.; Kim, C.; Han, S. S. Pattern Learning Electronic Density of States. *Sci. Rep.* **2019**, 9, 5879.

(42) Curtarolo, S.; Morgan, D.; Persson, K. A.; Rodgers, J.; Ceder, G. Predicting crystal structures with data mining of quantum calculations. *Phys. Rev. Lett.* **2003**, 91, 135503.

(43) Fischer, C. C.; Tibbetts, K. J.; Morgan, D.; Ceder, G. Predicting crystal structure by merging data mining with quantum mechanics. *Nat. Mater.* **2006**, 5, 641–646.

(44) Hautier, G.; Fischer, C. C.; Jain, A.; Mueller, T.; Ceder, G. Finding Nature's Missing Ternary Oxide Compounds Using Machine Learning and Density Functional Theory. *Chem. Mater.* **2010**, 22, 3762–3767.

(45) Kong, C. S.; Luo, W.; Arapan, S.; Villars, P.; Iwata, S.; Ahuja, R.; Rajan, K. Information-theoretic approach for the discovery of design rules for crystal chemistry. *J. Chem. Inf. Model.* **2012**, 52, 1812–1820.

(46) Pilania, G.; Balachandran, P. V.; Gubernatis, J. E.; Lookman, T. Classification of ABO₃ perovskite solids: a machine learning study. *Acta Crystallogr., Sect. B: Struct. Sci., Cryst. Eng. Mater.* **2015**, 71, S07–S13.

(47) Oliynyk, A. O.; Adutwum, L. A.; Harynyuk, J. J.; Mar, A. Classifying Crystal Structures of Binary Compounds AB through Cluster Resolution Feature Selection and Support Vector Machine Analysis. *Chem. Mater.* **2016**, 28, 6672–6681.

(48) Goldsmith, B. R.; Boley, M.; Vreeken, J.; Scheffler, M.; Ghiringhelli, L. M. Uncovering structure-property relationships of materials by subgroup discovery. *New J. Phys.* **2017**, 19, 013031.

(49) Balachandran, P. V.; Young, J.; Lookman, T.; Rondonelli, J. M. Learning from data to design functional materials without inversion symmetry. *Nat. Commun.* **2017**, 8, 14282.

(50) Schmidt, J.; Shi, J.; Borlido, P.; Chen, L.; Botti, S.; Marques, M. A. L. Predicting the Thermodynamic Stability of Solids Combining Density Functional Theory and Machine Learning. *Chem. Mater.* **2017**, 29, S090–S103.

(51) Seko, A.; Hayashi, H.; Kashima, H.; Tanaka, I. Matrix- and tensor-based recommender systems for the discovery of currently unknown inorganic compounds. *Physical Review Materials* **2018**, 2, 013805.

(52) Li, W.; Jacobs, R.; Morgan, D. Predicting the thermodynamic stability of perovskite oxides using machine learning models. *Comput. Mater. Sci.* **2018**, 150, 454–463.

(53) Isayev, O.; Oses, C.; Toher, C.; Gossett, E.; Curtarolo, S.; Tropsha, A. Universal fragment descriptors for predicting properties of inorganic crystals. *Nat. Commun.* **2017**, 8, 15679.

(54) Kauwe, S. K.; Graser, J.; Vazquez, A.; Sparks, T. D. Machine Learning Prediction of Heat Capacity for Solid Inorganics. *Integrating Materials and Manufacturing Innovation* **2018**, 7, 43–51.

(55) Carrete, J.; Li, W.; Mingo, N.; Wang, S.; Curtarolo, S. Finding Unprecedentedly Low-Thermal-Conductivity Half-Heusler Semiconductors via High-Throughput Materials Modeling. *Phys. Rev. X* **2014**, 4, 011019.

(56) Gaultois, M. W.; Oliynyk, A. O.; Mar, A.; Sparks, T. D.; Mulholland, G. J.; Meredig, B. Perspective: Web-based machine learning models for real-time screening of thermoelectric materials properties. *APL Mater.* **2016**, 4, 053213.

(57) Seko, A.; Hayashi, H.; Nakayama, K.; Takahashi, A.; Tanaka, I. Representation of compounds for machine-learning prediction of physical properties. *Phys. Rev. B: Condens. Matter Mater. Phys.* **2017**, 95, 144110.

(58) Furmanchuk, A.; Saal, J. E.; Doak, J. W.; Olson, G. B.; Choudhary, A.; Agrawal, A. Prediction of seebeck coefficient for compounds without restriction to fixed stoichiometry: A machine learning approach. *J. Comput. Chem.* **2018**, 39, 191–202.

(59) Wei, L.; Xu, X.; Gurudayal; Bullock, J.; Ager, J. W. Machine Learning Optimization of p-Type Transparent Conducting Films. *Chem. Mater.* **2019**, 31, 7340–7350.

(60) Davies, D. W.; Butler, K. T.; Walsh, A. Data-Driven Discovery of Photoactive Quaternary Oxides Using First-Principles Machine Learning. *Chem. Mater.* **2019**, 31, 7221–7230.

(61) Sendek, A. D.; Yang, Q.; Cubuk, E. D.; Duerloo, K.-A. N.; Cui, Y.; Reed, E. J. Holistic computational structure screening of more than 12000 candidates for solid lithium-ion conductor materials. *Energy Environ. Sci.* **2017**, 10, 306–320.

(62) Ahmad, Z.; Xie, T.; Maheshwari, C.; Grossman, J. C.; Viswanathan, V. Machine Learning Enabled Computational Screening of Inorganic Solid Electrolytes for Suppression of Dendrite Formation in Lithium Metal Anodes. *ACS Cent. Sci.* **2018**, 4, 996–1006.

(63) Sendek, A. D.; Cubuk, E. D.; Antoniak, E. R.; Cheon, G.; Cui, Y.; Reed, E. J. Machine Learning-Assisted Discovery of Solid Li-Ion Conducting Materials. *Chem. Mater.* **2019**, 31, 342–352.

(64) Bobbitt, N. S.; Snurr, R. Q. Molecular modelling and machine learning for high-throughput screening of metal-organic frameworks for hydrogen storage. *Mol. Simul.* **2019**, 45, 1069–1081.

(65) Gu, G. H.; Noh, J.; Kim, I.; Jung, Y. Machine learning for renewable energy materials. *J. Mater. Chem. A* **2019**, 7, 17096–17117.

(66) Seh, Z. W.; Kibsgaard, J.; Dickens, C. F.; Chorkendorff, I.; Nørskov, J. K.; Jaramillo, T. F. Combining theory and experiment in electrocatalysis: Insights into materials design. *Science* **2017**, 355, No. eaad4998.

(67) Ulissi, Z. W.; Medford, A. J.; Bligaard, T.; Nørskov, J. K. To address surface reaction network complexity using scaling relations machine learning and DFT calculations. *Nat. Commun.* **2017**, 8, 14621.

(68) Kitchin, J. R. Machine learning in catalysis. *Nature Catalysis* **2018**, 1, 230–232.

(69) Hansen, M. H.; Torres, J. A. G.; Jennings, P. C.; Wang, Z.; Boes, J. R.; Mamun, O. G.; Bligaard, T. An Atomistic Machine Learning Package for Surface Science and Catalysis. *arXiv*, 2019. <https://arxiv.org/pdf/1904.00904v1>, accessed May 5, 2020.

(70) Schlexer Lamoureux, P.; Winther, K. T.; Garrido Torres, J. A.; Streibel, V.; Zhao, M.; Bajdich, M.; Abild-Pedersen, F.; Bligaard, T. Machine Learning for Computational Heterogeneous Catalysis. *ChemCatChem* **2019**, 11, 3581–3601.

(71) Masood, H.; Toe, C. Y.; Teoh, W. Y.; Sethu, V.; Amal, R. Machine Learning for Accelerated Discovery of Solar Photocatalysts. *ACS Catal.* **2019**, 9, 11774–11787.

(72) Gaultois, M. W.; Sparks, T. D.; Borg, C. K. H.; Seshadri, R.; Bonificio, W. D.; Clarke, D. R. Data-Driven Review of Thermoelectric Materials: Performance and Resource Considerations. *Chem. Mater.* **2013**, 25, 2911–2920.

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

- (73) Sparks, T. D.; Gaultois, M. W.; Oliynyk, A. O.; Brgoch, J.; Meredig, B. Data mining our way to the next generation of thermoelectrics. *Scr. Mater.* **2016**, *111*, 10–15.
- (74) Stanev, V.; Oses, C.; Kusne, A. G.; Rodriguez, E.; Paglione, J.; Curtarolo, S.; Takeuchi, I. Machine learning modeling of superconducting critical temperature. *npj Computational Materials* **2018**, *4*, 29.
- (75) Meredig, B.; Antono, E.; Church, C.; Hutchinson, M.; Ling, J.; Paradiso, S.; Blaiszik, B.; Foster, L.; Gibbons, B.; Hatrick-Simpers, J.; Mehta, A.; Ward, L. Can machine learning identify the next high-temperature superconductor? Examining extrapolation performance for materials discovery. *Molecular Systems Design & Engineering* **2018**, *3*, 819–825.
- (76) Konno, T.; Kurokawa, H.; Nabeshima, F.; Sakishita, Y.; Ogawa, R.; Hosako, I.; Maeda, A. Deep Learning Model for Finding New Superconductors. *arXiv*, 2018. <http://arxiv.org/pdf/1812.01995v3>, accessed May 5, 2020.
- (77) Hamidieh, K. A data-driven statistical model for predicting the critical temperature of a superconductor. *Comput. Mater. Sci.* **2018**, *154*, 346–354.
- (78) Dan, Y.; Dong, R.; Cao, Z.; Li, X.; Niu, C.; Li, S.; Hu, J. Computational Prediction of Critical Temperatures of Superconductors Based on Convolutional Gradient Boosting Decision Trees. *IEEE Access* **2020**, *8*, 57868–57878.
- (79) Matsumoto, K.; Horide, T. An acceleration search method of higher T_c superconductors by a machine learning algorithm. *Appl. Phys. Express* **2019**, *12*, 073003.
- (80) Roter, B.; Dordevic, S. V. Predicting new superconductors and their critical temperatures using unsupervised machine learning. *arXiv*, 2020. <http://arxiv.org/pdf/2002.07266v1>, accessed May 5, 2020.
- (81) Wen, C.; Zhang, Y.; Wang, C.; Xue, D.; Bai, Y.; Antonov, S.; Dai, L.; Lookman, T.; Su, Y. Machine learning assisted design of high entropy alloys with desired property. *Acta Mater.* **2019**, *170*, 109–117.
- (82) Chang, Y.-J.; Jui, C.-Y.; Lee, W.-J.; Yeh, A.-C. Prediction of the Composition and Hardness of High-Entropy Alloys by Machine Learning. *JOM* **2019**, *71*, 3433–3442.
- (83) Ren, F.; Ward, L.; Williams, T.; Laws, K. J.; Wolverton, C.; Hatrick-Simpers, J.; Mehta, A. Accelerated discovery of metallic glasses through iteration of machine learning and high-throughput experiments. *Science Advances* **2018**, *4*, No. eaq1566.
- (84) Ward, L.; O’Keeffe, S. C.; Stevick, J.; Jelbert, G. R.; Aykol, M.; Wolverton, C. A machine learning approach for engineering bulk metallic glass alloys. *Acta Mater.* **2018**, *159*, 102–111.
- (85) Jain, A.; Ong, S. P.; Hautier, G.; Chen, W.; Richards, W. D.; Dacek, S.; Cholia, S.; Gunter, D.; Skinner, D.; Ceder, G.; Persson, K. A. Commentary: The Materials Project: A materials genome approach to accelerating materials innovation. *APL Mater.* **2013**, *1*, 011002.
- (86) Saal, J. E.; Kirklin, S.; Aykol, M.; Meredig, B.; Wolverton, C. Materials Design and Discovery with High-Throughput Density Functional Theory: The Open Quantum Materials Database (OQMD). *JOM* **2013**, *65*, 1501–1509.
- (87) Curtarolo, S.; Setyawan, W.; Wang, S.; Xue, J.; Yang, K.; Taylor, R. H.; Nelson, L. J.; Hart, G. L.; Sanvito, S.; Buongiorno-Nardelli, M.; Mingo, N.; Levy, O. AFLOWLIB.ORG: A distributed materials properties repository from high-throughput ab initio calculations. *Comput. Mater. Sci.* **2012**, *58*, 227–235.
- (88) Draxl, C.; Scheffler, M. The NOMAD laboratory: from data sharing to artificial intelligence. *Journal of Physics: Materials* **2019**, *2*, 036001.
- (89) Open Materials Database. <http://openmaterialsdb.se/index.php>, accessed May 5, 2020.
- (90) Citrine Informatics: The AI Platform for Materials Development. <https://citrine.io/>, accessed May 5, 2020.
- (91) Materials Platform for Data Science (MPDS). <https://mpds.io/>, accessed May 5, 2020.
- (92) Huber, S. P.; et al. AiiDA 1.0, a scalable computational infrastructure for automated reproducible workflows and data provenance. *arXiv*, 2020. <http://arxiv.org/pdf/2003.12476v1>, accessed May 5, 2020.
- (93) Talirz, L.; et al. Materials Cloud, a platform for open computational science. *arXiv*, 2020. <http://arxiv.org/pdf/2003.12510v1>, accessed May 5, 2020.
- (94) Deml, A.; Lany, S.; Peng, H.; Stevanovic, V.; Yan, J.; Zawadzki, P.; Graf, P.; Sorensen, H.; Sullivan, S. NREL MatDB. <https://materials.nrel.gov/>, accessed May 5, 2020.
- (95) National Institute of Standards and Technology (NIST). NIST TRC Alloy Data. 2017. <https://www.nist.gov/mml/acmd/trc/nist-alloy-data>, accessed May 5, 2020.
- (96) National Institute of Standards and Technology (NIST). NIST TRC ThermoData Engine. 2005. <https://www.nist.gov/mml/acmd/trc/thermodata-engine>, accessed May 5, 2020.
- (97) National Institute of Standards and Technology (NIST). NIST JARVIS-DFT Database. 2017. <https://www.nist.gov/programs-projects/jarvis-dft>, accessed May 5, 2020.
- (98) National Institute of Standards and Technology (NIST). NIST JARVIS-ML Database. 2019. <https://www.nist.gov/programs-projects/jarvis-ml>, accessed May 5, 2020.
- (99) MatWeb. <http://www.matweb.com/index.aspx>, accessed May 5, 2020.
- (100) Total Materia. <https://www.totalmateria.com/>, accessed May 5, 2020.
- (101) Ansys Granta MaterialUniverse. <https://grantadesign.com/>, accessed May 5, 2020.
- (102) MATDAT. <https://www.matdat.com/>, accessed May 5, 2020.
- (103) Groom, C. R.; Bruno, I. J.; Lightfoot, M. P.; Ward, S. C. The Cambridge Structural Database. *Acta Crystallogr., Sect. B: Struct. Sci., Cryst. Eng. Mater.* **2016**, *72*, 171–179.
- (104) Hellenbrandt, M. The Inorganic Crystal Structure Database (ICSD)—Present and Future. *Crystallogr. Rev.* **2004**, *10*, 17–22.
- (105) Pearson’s Crystal Data: Crystal Structure Database for Inorganic Compounds. <https://www.crystalimpact.com/pcd/Default.htm>, accessed May 5, 2020.
- (106) Gates-Rector, S.; Blanton, T. The Powder Diffraction File: a quality materials characterization database. *Powder Diff.* **2019**, *34*, 352–360.
- (107) Gražulis, S.; Chateigner, D.; Downs, R. T.; Yokochi, A. F. T.; Quirós, M.; Lutterotti, L.; Manakova, E.; Butkus, J.; Moeck, P.; Le Bail, A. Crystallography Open Database – an open-access collection of crystal structures. *J. Appl. Crystallogr.* **2009**, *42*, 726–729.
- (108) Pauling File. <https://paulingfile.com/>, accessed May 5, 2020.
- (109) White, P. S.; Rodgers, J. R.; Le Page, Y. CRYSTMET: a database of the structures and powder patterns of metals and intermetallics. *Acta Crystallogr., Sect. B: Struct. Sci.* **2002**, *58*, 343–348.
- (110) van der Maaten, L.; Hinton, G. Visualizing Data using t-SNE. *Journal of Machine Learning Research* **2008**, *9*, 2579–2605.
- (111) McInnes, L.; Healy, J.; Saul, N.; Großberger, L. UMAP: Uniform Manifold Approximation and Projection. *Journal of Open Source Software* **2018**, *3*, 861.
- (112) Kauwe, S. K.; Yang, Y.; Sparks, T. D. Visualization Tool for Atomic Models (VITAL): A Simple Visualization Tool for Materials Predictions. *ChemRxiv*, 2019. DOI: 10.26434/chemrxiv.9782375, accessed May 5, 2020.
- (113) Git. <https://git-scm.com/>, accessed May 5, 2020.
- (114) Mercurial. <https://www.mercurial-scm.org/>, accessed May 5, 2020.
- (115) Apache® Subversion®. <https://subversion.apache.org/>, accessed May 5, 2020.
- (116) Cawley, G. C.; Talbot, N. L. On Over-Fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation. *Journal of Machine Learning Research* **2010**, *11*, 2079–2107.
- (117) Pedregosa, F.; et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **2011**, *12*, 2825–2830.
- (118) Paszke, A.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv*, 2019. <http://arxiv.org/pdf/1912.01703v1>, accessed May 5, 2020.

3.2 PUBLICATION 1: BESTPRACTICES

- (119) Abadi, M.; et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. <https://www.tensorflow.org/>, accessed May 5, 2020.
- (120) Graser, J.; Kauwe, S. K.; Sparks, T. D. Machine Learning and Energy Minimization Approaches for Crystal Structure Predictions: A Review and New Horizons. *Chem. Mater.* **2018**, *30*, 3601–3612.
- (121) Choudhary, K.; DeCost, B.; Tavazza, F. Machine learning with force-field-inspired descriptors for materials: Fast screening and mapping energy landscape. *Physical Review Materials* **2018**, *2*, 083801.
- (122) Jha, D.; Ward, L.; Yang, Z.; Wolverton, C.; Foster, I.; Liao, W.-K.; Choudhary, A.; Agrawal, A. IRNet: A General Purpose Deep Residual Regression Framework for Materials Discovery. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining – KDD '19*; ACM: New York, NY, U.S.A., 2019; pp 2385–2393.
- (123) Juszczak, P.; Tax, D. M.; Duin, R. P. Feature scaling in support vector data description. *Proceedings of the Eighth Annual Conference of the Advanced School for Computing and Imaging*; AAAI: 2002; pp 95–102.
- (124) Ba, J. L.; Kiros, J. R.; Hinton, G. E. Layer Normalization. *arXiv*, 2016. <http://arxiv.org/pdf/1607.06450v1>, accessed May 5, 2020.
- (125) Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv*, 2015. <http://arxiv.org/pdf/1502.03167v3>, accessed May 5, 2020.
- (126) Mohamad, I. B.; Usman, D. Standardization and Its Effects on k-Means Clustering Algorithm. *Res. J. Appl. Sci., Eng. Technol.* **2013**, *6*, 3299–3303.
- (127) Head, M. L.; Holman, L.; Lanfear, R.; Kahn, A. T.; Jennions, M. D. The extent and consequences of p-hacking in science. *PLoS Biol.* **2015**, *13*, No. e1002106.
- (128) Ward, L.; et al. Matminer: An open source toolkit for materials data mining. *Comput. Mater. Sci.* **2018**, *152*, 60–69.
- (129) Olson, R. S.; Bartley, N.; Urbanowicz, R. J.; Moore, J. H. Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO '16*; ACM: New York, NY, USA, 2016; pp 485–492.
- (130) *Automatminer*. <https://github.com/hackingmaterials/automatminer>, accessed May 5, 2020.
- (131) Dunn, A.; Wang, Q.; Ganose, A.; Dopp, D.; Jain, A. Benchmarking Materials Property Prediction Methods: The Matbench Test Set and Automatminer Reference Algorithm. *arXiv*, 2020. <http://arxiv.org/pdf/2005.00707v1>, accessed May 5, 2020.
- (132) Barnes, N. Publish your computer code: it is good enough. *Nature* **2010**, *467*, 753.
- (133) Chard, R.; Li, Z.; Chard, K.; Ward, L.; Babuji, Y.; Woodard, A.; Tuecke, S.; Blaiszik, B.; Franklin, M. J.; Foster, I. DLHub: Model and Data Serving for Science. *arXiv*, 2018. <http://arxiv.org/pdf/1811.11213v1>, accessed May 5, 2020.
- (134) Docker. <https://www.docker.com/>, accessed May 5, 2020.
- (135) Clement, C. L.; Kauwe, S. K.; Sparks, T. D. Benchmark AFLOW Data Sets for Machine Learning. *Integrating Materials and Manufacturing Innovation* **2020**, DOI: 10.1007/s40192-020-00174-4.
- (136) Clement, C. L.; Kauwe, S. K.; Sparks, T. D. *Benchmark AFLOW Data Sets for Machine Learning*. figshare: 2020. DOI: 10.6084/m9.figshare.11954742, accessed May 5, 2020.

Supplementary Information for the article “Machine
Learning for Materials Scientists:
An Introductory Guide toward Best Practices”

Authors: Anthony Yu-Tung Wang, Ryan J. Murdock, Steven K. Kauwe, Anton O.
Oliyynyk, Aleksander Gurlo, Jakoah Brgoch, Kristin A. Persson, and Taylor D. Sparks

DOI: [10.1021/acs.chemmater.0c01907](https://doi.org/10.1021/acs.chemmater.0c01907)

Overview of Notebooks

These notebooks are included to illustrate a hypothetical Machine Learning project created following best practices.

The goal of this ML project is to predict the heat capacity of inorganic materials given the chemical composition and condition (the measurement temperature). We will use both classical ML models as well as neural networks.

To do this, we must:

1. Clean and process our dataset, removing obviously erroneous or empty values.
2. Partition our data into train, validation, and test splits.
3. Featurize our data, turning the chemical formulae into CBFVs.
4. Train models on our data and assess the predictive power of the models.
5. Compare the performance of the models fairly and reproducibly.
6. Visualize the prediction results of the models.
7. Share our models and enable others to reproduce your work and aid collaboration.

If you require more information about how to use Jupyter notebooks, you can consult:

- The main README file inside this repository: <https://github.com/anthony-wang/BestPractices/blob/master/README.md> (<https://github.com/anthony-wang/BestPractices/blob/master/README.md>)
- The official Jupyter Notebook documentation: <https://jupyter-notebook.readthedocs.io/en/stable/notebook.html> (<https://jupyter-notebook.readthedocs.io/en/stable/notebook.html>)

To read the main publication for which these notebooks are made, please see:

Wang, Anthony Yu-Tung; Murdock, Ryan J.; Kauwe, Steven K.; Oliynyk, Anton O.; Gurlo, Aleksander; Brgoch, Jakoah; Persson, Kristin A.; Sparks, Taylor D., [Machine Learning for Materials Scientists: An Introductory Guide toward Best Practices](https://doi.org/10.1021/acs.chemmater.0c01907) (<https://doi.org/10.1021/acs.chemmater.0c01907>), *Chemistry of Materials* **Just Accepted Manuscript**, 2020. DOI: [10.1021/acs.chemmater.0c01907](https://doi.org/10.1021/acs.chemmater.0c01907) (<https://doi.org/10.1021/acs.chemmater.0c01907>)

Please also consider citing the work if you choose to adopt or adapt the methods and concepts shown in these notebooks or in the publication:

```
@article{Wang2020bestpractices,
  author = {Wang, Anthony Yu-Tung and Murdock, Ryan J. and Kauwe, Steven K. and Oliynyk, Anton O. and Gurlo, Aleksander and Brgoch, Jakoah and Persson, Kristin A. and Sparks, Taylor D.},
  date = {2020},
  title = {Machine Learning for Materials Scientists: An Introductory Guide toward Best Practices},
  issn = {0897-4756},
  journal = {Chemistry of Materials},
  url = {https://doi.org/10.1021/acs.chemmater.0c01907},
  doi = {10.1021/acs.chemmater.0c01907}
}
```

Check that libraries are installed

This notebook checks to see if you have the correct version of Python as well as all necessary libraries installed.

Check the [main README file \(https://github.com/anthony-wang/BestPractices/blob/master/README.md\)](https://github.com/anthony-wang/BestPractices/blob/master/README.md) for instructions if anything is missing.

3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [1]: from __future__ import print_function
from distutils.version import LooseVersion as Version
import sys

try:
    import curses
    curses.setupterm()
    assert curses.tigetnum("colors") > 2
    OK = "\x1b[1;32m[ OK ]\x1b[0m" % (30 + curses.COLOR_GREEN)
    FAIL = "\x1b[1;31m[FAIL]\x1b[0m" % (30 + curses.COLOR_RED)
except:
    OK = '[ OK ]'
    FAIL = '[FAIL]'

try:
    import importlib
except ImportError:
    print(FAIL, "Python version 3.4 is required,"
          " but %s is installed." % sys.version)

def import_version(pkg, min_ver):
    mod = None
    try:
        mod = importlib.import_module(pkg)
        if pkg in {'PIL'}:
            ver = mod.VERSION
        else:
            ver = mod.__version__
        if Version(ver) < min_ver:
            print(FAIL, "%s version %s or higher required, but %s installed."
                  % (lib, min_ver, ver))
        else:
            print(OK, '%s version %s' % (pkg, ver))
    except ImportError as imp_err_msg:
        print(FAIL, 'Error in importing %s: %s' % (pkg, imp_err_msg))
    except AttributeError as att_err_msg:
        print(FAIL, 'Error in reading attribute of %s: %s' % (pkg, att_err_msg))
    return mod

# first check the python version
print('Using python in', sys.prefix)
print(sys.version)
pyversion = Version(sys.version)
if pyversion >= "3":
    if pyversion < "3.7":
        print(FAIL, "Python version > 3.7 is required,"
              " but %s is installed.\n" % sys.version)
elif pyversion < "3":
    print(FAIL, "Python version > 3.7 is required,"
          " but %s is installed.\n" % sys.version)
else:
    print(FAIL, "Unknown Python version: %s\n" % sys.version)

requirements = {'numpy': '1.18.0',
                'pandas': '1.0.0',
```

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
'pandas_profiling': '2.4.0',  
'matplotlib': '3.2.0',  
'seaborn': '0.10.0',  
'sklearn': '0.22.0',  
'scipy': '1.4.0',  
'tqdm': '4.43.0',  
'jupyter_client': '6.0.0',  
'ipywidgets': '7.5.0',  
'torch': '1.3.0',}  
  
# now check the dependencies  
for lib, required_version in list(requirements.items()):  
    import_version(lib, required_version)
```

```
Using python in C:\Users\Anthony\Anaconda3\envs\bestpractices  
3.7.6 | packaged by conda-forge | (default, Mar 5 2020, 14:47:50) [MSC v.191  
6 64 bit (AMD64)]  
[ OK ] numpy version 1.18.1  
[ OK ] pandas version 1.0.2  
[ OK ] pandas_profiling version 2.4.0  
[ OK ] matplotlib version 3.2.0  
[ OK ] seaborn version 0.10.0  
[ OK ] sklearn version 0.22.2.post1  
[ OK ] scipy version 1.4.1  
[ OK ] tqdm version 4.43.0  
[ OK ] jupyter_client version 6.1.2  
[ OK ] ipywidgets version 7.5.1  
[ OK ] torch version 1.3.1
```

Data loading, cleanup and processing

The first step to a ML project is to obtain the dataset you will be working with. There are many repositories for materials science-specific data (whether online or offline)---consult the accompanying paper for a list of the more commonly used ones.

Once you have identified the repository and dataset you will use for your project, you will have to download it to your local machine, or establish a way to reliably access the dataset. Consult the documentation of the repository for how to do this.

For this tutorial, we have collected heat capacity (C_p) data from the [NIST-JANAF Thermochemical Tables](https://doi.org/10.18434/T42S31) (<https://doi.org/10.18434/T42S31>).

```
In [1]: import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format='retina'

from pandas_profiling import ProfileReport
```

Load data

Using Pandas, we read in the dataset into a DataFrame.

We also print the shape of the DataFrame, which indicates the number of rows and columns in this dataset.

```
In [2]: PATH = os.getcwd()
data_path = os.path.join(PATH, '../data/cp_data_demo.csv')

df = pd.read_csv(data_path)
print(f'Original DataFrame shape: {df.shape}')

Original DataFrame shape: (4583, 3)
```

This means that our input dataset has 4583 data samples, each with 3 variables.

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

Examine the data

We examine some rows and look at the data's basic statistics.

We see that the dataset contains information about the formula, measurement condition (in this case, temperature in K), and the target property, heat capacity (in J/(mol * K)).

```
In [3]: df.head(10)
```

Out[3]:

	FORMULA	CONDITION: Temperature (K)	PROPERTY: Heat Capacity (J/mol K)
0	B2O3	1400.0	134.306
1	B2O3	1300.0	131.294
2	B2O3	1200.0	128.072
3	B2O3	1100.0	124.516
4	B2O3	1000.0	120.625
5	B2O3	900.0	116.190
6	B2O3	800.0	111.169
7	B2O3	723.0	106.692
8	B2O3	700.0	105.228
9	B2O3	600.0	98.115

First thing you should notice: we have many observations of the same compound (B2O3) but measured at different measurement conditions, resulting in a different property value.

We can get some simple summary statistics of the DataFrame by calling the `.describe()` method on the database.

3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [4]: df.describe()
```

Out[4]:

	CONDITION: Temperature (K)	PROPERTY: Heat Capacity (J/mol K)
count	4579.000000	4576.000000
mean	1170.920341	107.483627
std	741.254366	67.019055
min	-2000.000000	-102.215000
25%	600.000000	61.312500
50%	1000.000000	89.497000
75%	1600.000000	135.645000
max	4700.000000	494.967000

Using the `pandas-profiling` library, we can generate a more in-depth report of our starting dataset. Note that generating this profile report might take upwards of 20 seconds.

```
In [5]: profile = ProfileReport(df.copy(), title='Pandas Profiling Report of Cp dataset',
                                html={'style':{'full_width':True}})
        profile.to_widgets()
```

Report generated with [pandas-profiling \(https://github.com/pandas-profiling/pandas-profiling\)](https://github.com/pandas-profiling/pandas-profiling).

Notice a few things from the profile report:

- We have some missing cells in the dataset ("Overview" tab)
- We have some unrealistic Temperature and Heat Capacity values in the dataset ("Variables" tab)
- We have some missing Temperature, Formula and Heat Capacity values in the dataset ("Variables" tab)

Also notice that on the "Overview" tab, there is the following warning: `FORMULA` has a high cardinality: 245 distinct values.

Cardinality is the number of distinct values in a column of a table, relative to the number of rows in the table. In our dataset, we have a total of 4583 data observations, but only 245 distinct formulae. We will have to keep this in mind later, when we process and split the dataset.

Rename the column names for brevity

```
In [6]: df.columns
```

```
Out[6]: Index(['FORMULA', 'CONDITION: Temperature (K)',
               'PROPERTY: Heat Capacity (J/mol K)'],
              dtype='object')
```

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [7]: rename_dict = {'FORMULA': 'formula',  
                      'CONDITION: Temperature (K)': 'T',  
                      'PROPERTY: Heat Capacity (J/mol K)': 'Cp'}  
df = df.rename(columns=rename_dict)  
df.columns
```

```
Out[7]: Index(['formula', 'T', 'Cp'], dtype='object')
```

Check for and remove NaN values

Here we can use the built-in Pandas methods to check for NaN values in the dataset, which are missing values. We then remove the dataset rows which contain NaN values.

```
In [8]: # Check for NaNs in the respective dataset columns, and get the indices  
df2 = df.copy()  
bool_nans_formula = df2['formula'].isnull()  
bool_nans_T = df2['T'].isnull()  
bool_nans_Cp = df2['Cp'].isnull()  
  
# Drop the rows of the DataFrame which contain NaNs  
df2 = df2.drop(df2.loc[bool_nans_formula].index, axis=0)  
df2 = df2.drop(df2.loc[bool_nans_T].index, axis=0)  
df2 = df2.drop(df2.loc[bool_nans_Cp].index, axis=0)  
  
print(f'DataFrame shape before dropping NaNs: {df.shape}')  
print(f'DataFrame shape after dropping NaNs: {df2.shape}')
```

```
DataFrame shape before dropping NaNs: (4583, 3)  
DataFrame shape after dropping NaNs: (4570, 3)
```

Pandas also includes the convenient built-in method `.dropna()` to check for and remove NaNs in-place:

```
In [9]: df3 = df.copy()  
df3 = df3.dropna(axis=0, how='any')  
  
print(f'DataFrame shape before dropping NaNs: {df.shape}')  
print(f'DataFrame shape after dropping NaNs: {df3.shape}')  
  
df = df3.copy()
```

```
DataFrame shape before dropping NaNs: (4583, 3)  
DataFrame shape after dropping NaNs: (4570, 3)
```

Check for and remove unrealistic values

In some cases, you might also get data values that simply don't make sense. For our case, this could be negative values in the temperature or heat capacity values.

3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [10]: bool_invalid_T = df['T'] < 0
         bool_invalid_Cp = df['Cp'] < 0

         df = df.drop(df.loc[bool_invalid_T].index, axis=0)
         df = df.drop(df.loc[bool_invalid_Cp].index, axis=0)

         print(f'Cleaned DataFrame shape: {df.shape}')

Cleaned DataFrame shape: (4564, 3)
```

Save cleaned data to csv

Finally, after cleaning and processing the data, you can save it to disk in a cleaned state for you to use later.

Pandas allows us to save our data as a comma separated value .csv file.

```
In [11]: out_path = os.path.join(PATH, '../data/cp_data_cleaned.csv')
         df.to_csv(out_path, index=False)
```

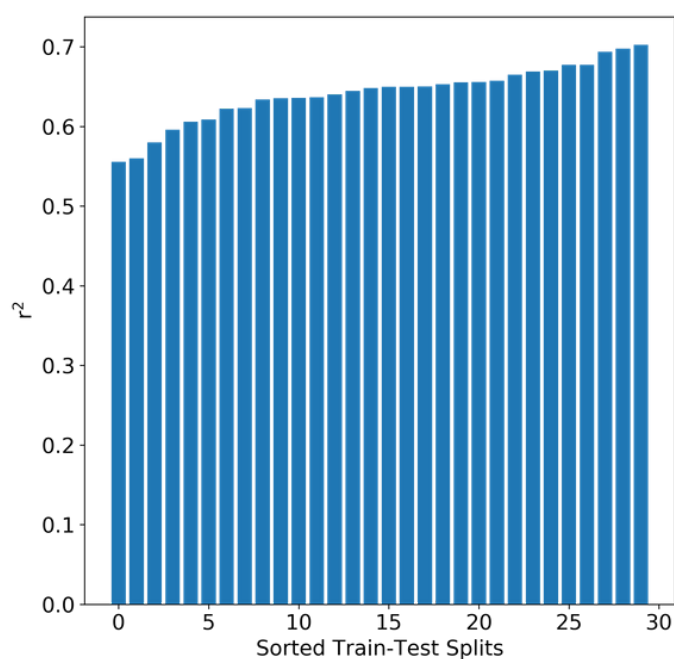
Note, your data can be saved in other file formats (such as hdf5) or in databases (such as SQL), but we will not go into the details of these formats.

Typically, the amount of data you can gather for your ML project isn't large enough to warrant these approaches.

Splitting data into the train/validation/test dataset

It is important to split your full dataset into train/validation/test datasets, and reliably use the same datasets for your modeling tasks later.

Using different train/validation/test splits can dramatically affect your model performance (as seen here by the variance in r^2 scores for 30 models which have been trained on 30 different dataset splits) [1]:



[1]: C. Clement, S. K. Kauwe, T. D. Sparks, Benchmark AFLOW Data Sets for Machine Learning, figshare 2020, DOI: [10.6084/m9.figshare.11954742](https://doi.org/10.6084/m9.figshare.11954742) (<https://dx.doi.org/10.6084/m9.figshare.11954742>).

3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [1]: import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format='retina'

from sklearn.model_selection import train_test_split

# Set a random seed to ensure reproducibility across runs
RNG_SEED = 42
np.random.seed(seed=RNG_SEED)
```

Load the pre-processed dataset

We will start with the processed dataset that we saved from the last notebook.

```
In [2]: PATH = os.getcwd()
data_path = os.path.join(PATH, '../data/cp_data_cleaned.csv')

df = pd.read_csv(data_path)
print(f'Full DataFrame shape: {df.shape}')

Full DataFrame shape: (4564, 3)
```

```
In [3]: df.head(10)
```

Out[3]:

	formula	T	Cp
0	B2O3	1400.0	134.306
1	B2O3	1300.0	131.294
2	B2O3	1200.0	128.072
3	B2O3	1100.0	124.516
4	B2O3	1000.0	120.625
5	B2O3	900.0	116.190
6	B2O3	800.0	111.169
7	B2O3	723.0	106.692
8	B2O3	700.0	105.228
9	B2O3	600.0	98.115

Separate the DataFrame into your input variables (X) and target variables (y)

The X will be used as the input data, and y will be used as the prediction targets for your ML model.

If your target variables are discrete (such as `metal / non-metal` or types of crystal structures), then you will be performing a classification task. In our case, since our target variables are continuous values (heat capacity), we are performing a regression task.

```
In [4]: X = df[['formula', 'T']]
        y = df['Cp']

        print(f'Shape of X: {X.shape}')
        print(f'Shape of y: {y.shape}')

Shape of X: (4564, 2)
Shape of y: (4564,)
```

Splitting data (and a word of caution)

Normally, we could simply split the data with a simple `sklearn` function

The scikit-learn `train_test_split` function randomly splits a dataset into train and test datasets. Typically, you can use `train_test_split` to first split your data into "train" and "test" datasets, and then use the function again to split your "train" data into "train" and "validation" dataset splits.

As a rule of thumb, you can roughly aim for the following dataset proportions when splitting your data:

	train split	validation split	test split
proportion of original dataset	50% to 70%	20% to 30%	10% to 20%

If you have copious amounts of data, it may suffice to train your models on just 50% of the data; that way, you have a larger amount of data samples to validate and to test with. If you however have a smaller dataset and thus very few training samples for your models, you may wish to increase your proportion of training data during dataset splitting.

```
In [5]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=RNG_SEED)

        print(X_train.shape)
        print(X_test.shape)

(3651, 2)
(913, 2)
```

But wait, what's wrong here?

We have to make sure that our dataset splits contain mutually exclusive formulae (e.g., all the data samples associated with "Al₂O₃" is *either* in the train, validation, or test dataset, but *not in multiple*)!

```
In [6]: num_rows = len(X_train)
print(f'There are in total {num_rows} rows in the X_train DataFrame.')

num_unique_formulae = len(X_train['formula'].unique())
print(f'But there are only {num_unique_formulae} unique formulae!\n')

print('Unique formulae and their number of occurrences in the X_train DataFrame:')
print(X_train['formula'].value_counts(), '\n')
print('Unique formulae and their number of occurrences in the X_test DataFrame:')
print(X_test['formula'].value_counts())
```

There are in total 3651 rows in the X_train DataFrame.
But there are only 244 unique formulae!

Unique formulae and their number of occurrences in the X_train DataFrame:

```
W1      40
N1Ti1   38
N1Zr1   33
B1Ti1   33
O2Zr1   30
..
I4Si1    4
N2O4     3
K1        2
Hg101    1
I4Ti1    1
```

Name: formula, Length: 244, dtype: int64

Unique formulae and their number of occurrences in the X_test DataFrame:

```
Ca1S1    10
N0.465V1 10
Be204Si1 10
O3W1      9
W1         9
..
H2Mg1     1
O2Pb1     1
Fe1H3O3   1
Hf1        1
I1K1       1
```

Name: formula, Length: 229, dtype: int64

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

There are in total 3651 rows in the `X_train` DataFrame. But there are only 244 unique formulae! In fact, you will see that the same formulae are often present in the `X_train` and `X_test` DataFrames!

That's not good, because now we have instances of the same chemical compound appearing in *both* the training and test data. Which means the model can cheat and in essence just memorize the training data, and during testing, look up the nearby values present in the training data!

So how do we mitigate this?

Be aware of leaking data between datasets

We have to first group the data by chemical formula, then split the data according to the chemical formulae. That way, all data points associated with each formula are either in the training dataset or in the test dataset, *but not in both at the same time*.

Splitting data, cautiously (manually)

First we get a list of all of the unique formulae in the dataset.

3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [7]: unique_formulae = X['formula'].unique()
print(f'{len(unique_formulae)} unique formulae:\n{unique_formulae}')
```

244 unique formulae:

```
['B2O3' 'BeI2' 'Be1F3Li1' 'Al1Cl4K1' 'Al2Be1O4' 'B2H4O4' 'B2Mg1' 'Be1F2'
'B1H4Na1' 'Br2Ca1' 'Al1N1' 'Al1Cl6Na3' 'Ba1H2O2' 'Al1Br3' 'Br3Zr1'
'Br2Ti1' 'B1Ti1' 'Be2O4Si1' 'Br2Pb1' 'Al1' 'Br2Hg2' 'B1H3O3' 'Br3Ti1'
'C1Cu1N1' 'B1' 'Al1F6Na3' 'Ca1H2O2' 'B2Be3O6' 'Al1Cl4Na1' 'Al1Cl6K3'
'C0.98Nb1' 'Br2Hg1' 'Al1Cl1O1' 'Cl1H4N1O4' 'Be1F4Li2' 'C1Mg1O3' 'Br1H4N1'
'Ca1I2' 'Al1F6Li3' 'Br4Mo1' 'Ba1' 'Br4Ti1' 'Ba1Br2' 'Be1O4S1' 'Ba1F2'
'Ba1I2' 'Cl2Fe1' 'C1K1N1' 'Be1H2O2' 'Cs1' 'Al1H4Li1' 'C1Be2' 'Cr1'
'Cs2O4S1' 'C11Cu1' 'Cu1F2' 'Al2O3' 'B1N1' 'Co1O4S1' 'Cu1O1' 'Br1Na1'
'Cr2O3' 'Cs1F1' 'Cr2N1' 'Cl1Li1' 'Fe0.877S1' 'Cl1Na1' 'F2Hg1' 'Fe1H2O2'
'Cs1H1O1' 'Br3Mo1' 'Br2Sr1' 'Cl2Hg2' 'Fe1O1' 'Co1' 'Cl1Cs1' 'Cu1H2O2'
'Al1Li1O2' 'Co1F2' 'Br2Fe1' 'Fe1I2' 'Ga1' 'Cl1Li1O4' 'Cl2Cu1' 'Fe0.947O1'
'Be1Cl2' 'Cl1K1' 'F1Na1' 'H3O4P1' 'Fe3O4' 'H1Na1O1' 'Fe2O12S3' 'H1Na1'
'Cl1Na1O4' 'B1F4K1' 'Cu1O4S1' 'H1Li1' 'F2H1K1' 'B1H4Li1' 'Hg1O1' 'Be3N2'
'Fe1' 'I2Mo1' 'Cu1F1' 'Cr1N1' 'Fe1H3O3' 'I1Li1' 'Al1I3' 'Fe1S1'
'Al2C19K3' 'I2Pb1' 'I4Zr1' 'Hg1I2' 'H4I1N1' 'Hf1' 'F2Hg2' 'I2Sr1'
'C1K2O3' 'C1N1Na1' 'H2O4W1' 'Ca1S1' 'K2O4S1' 'I2Mg1' 'Mg1O3Si1' 'Li3N1'
'I2Zr1' 'H2Mg1' 'I2Ti1' 'H1K1' 'Mg1O4W1' 'I4Ti1' 'H1K1O1' 'I2' 'Mn1'
'F1K1' 'Li2O3Si1' 'K2O1' 'Mg1O4S1' 'Al1Na1O2' 'Mo1O2.889' 'Mo1O2.750'
'N0.465V1' 'Mg2O4Ti1' 'K1O2' 'Mo1O3' 'C1Na2O3' 'K2S1' 'Mo1S2' 'Li2O3Ti1'
'I4Mo1' 'Ba1S1' 'Na2O3Si1' 'I3Mo1' 'Mg1S1' 'Cu2O5S1' 'K2O2' 'Mg1O3Ti1'
'Na2S2' 'I3Ti1' 'Li2O2' 'I3Zr1' 'Al2Mg1O4' 'N1Ti1' 'N1V1' 'Na1O2' 'Ni1S1'
'Na2O1' 'I4Si1' 'B1Li1O2' 'O1Ti1' 'H1Li1O1' 'Nb1O1' 'F2Mg1' 'Nb1' 'O3Ti2'
'Ca1' 'Nb1O2' 'O3Pb1Si1' 'O4Pb3' 'O3W1' 'O7Ti4' 'K1' 'O4V2' 'O2.90W1'
'Ca1Cl2' 'Pb1' 'Na2O5Si2' 'O5Ti3' 'O5V2' 'Mg3N2' 'Mg2O4Si1' 'Mo1O2.875'
'Br1K1' 'Br2Mo1' 'Cl1H4N1' 'Cu1' 'F1Li1' 'Fe1S2' 'H2O2Sr1' 'I1K1' 'I1Na1'
'K2O3Si1' 'Li2O4S1' 'Li2O5Si2' 'Mg1' 'Mg2Si1' 'Mo2S3' 'N1Zr1' 'N2O4'
'N4Si3' 'N5P3' 'Na2O2' 'Na2S1' 'Nb2O5' 'Ni1' 'Ni1S2' 'Ni3S2' 'Ni3S4'
'O10P4' 'O1Pb1' 'O1Sr1' 'O1V1' 'O2.72W1' 'O2.96W1' 'O2Pb1' 'O2Si1'
'O2Ti1' 'O2Zr1' 'O3V2' 'O4Pb2Si1' 'O4S1Zn1' 'O4Si1Zr1' 'P1' 'P4S3'
'Pb1S1' 'Rb1' 'S1' 'S1Sr1' 'Sr1' 'Ti1' 'V1' 'W1' 'Zn1' 'Zr1']
```

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [8]: # Set a random seed to ensure reproducibility across runs
np.random.seed(seed=RNG_SEED)

# Store a List of all unique formulae
all_formulae = unique_formulae.copy()

# Define the proportional size of the dataset split
val_size = 0.20
test_size = 0.10
train_size = 1 - val_size - test_size

# Calculate the number of samples in each dataset split
num_val_samples = int(round(val_size * len(unique_formulae)))
num_test_samples = int(round(test_size * len(unique_formulae)))
num_train_samples = int(round((1 - val_size - test_size) * len(unique_formulae)))

# Randomly choose the formulae for the validation dataset, and remove those from the unique formulae list
val_formulae = np.random.choice(all_formulae, size=num_val_samples, replace=False)
all_formulae = [f for f in all_formulae if f not in val_formulae]

# Randomly choose the formulae for the test dataset, and remove those from the unique formulae list
test_formulae = np.random.choice(all_formulae, size=num_test_samples, replace=False)
all_formulae = [f for f in all_formulae if f not in test_formulae]

# The remaining formulae will be used for the training dataset
train_formulae = all_formulae.copy()

print('Number of training formulae:', len(train_formulae))
print('Number of validation formulae:', len(val_formulae))
print('Number of testing formulae:', len(test_formulae))
```

```
Number of training formulae: 171
Number of validation formulae: 49
Number of testing formulae: 24
```

3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [9]: # Split the original dataset into the train/validation/test datasets using the
        formulae lists above
df_train = df[df['formula'].isin(train_formulae)]
df_val = df[df['formula'].isin(val_formulae)]
df_test = df[df['formula'].isin(test_formulae)]

print(f'train dataset shape: {df_train.shape}')
print(f'validation dataset shape: {df_val.shape}')
print(f'test dataset shape: {df_test.shape}\n')

print(df_train.head(), '\n')
print(df_val.head(), '\n')
print(df_test.head(), '\n')
```

```
train dataset shape: (3214, 3)
validation dataset shape: (980, 3)
test dataset shape: (370, 3)
```

	formula	T	Cp
0	B2O3	1400.0	134.306
1	B2O3	1300.0	131.294
2	B2O3	1200.0	128.072
3	B2O3	1100.0	124.516
4	B2O3	1000.0	120.625

	formula	T	Cp
82	B2Mg1	1900.0	92.242
83	B2Mg1	1800.0	90.249
84	B2Mg1	1700.0	88.162
85	B2Mg1	1600.0	85.981
86	B2Mg1	1500.0	83.643

	formula	T	Cp
192	Ba1H2O2	900.00	134.892
193	Ba1H2O2	800.00	130.834
194	Ba1H2O2	700.00	126.775
195	Ba1H2O2	681.15	126.022
196	Ba1H2O2	600.00	122.717

To be sure that we really only have mutually exclusive formulae within each of the datasets (e.g., all the data samples associated with "Al2O3" is *either* in the train, validation, or test dataset, but *not in multiple*), we can do the following to check:

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [10]: train_formulae = set(df_train['formula'].unique())
val_formulae = set(df_val['formula'].unique())
test_formulae = set(df_test['formula'].unique())

common_formulae1 = train_formulae.intersection(test_formulae)
common_formulae2 = train_formulae.intersection(val_formulae)
common_formulae3 = test_formulae.intersection(val_formulae)

print(f'# of common formulae in intersection 1: {len(common_formulae1)}; common formulae: {common_formulae1}')
print(f'# of common formulae in intersection 2: {len(common_formulae2)}; common formulae: {common_formulae2}')
print(f'# of common formulae in intersection 3: {len(common_formulae3)}; common formulae: {common_formulae3}')

# of common formulae in intersection 1: 0; common formulae: set()
# of common formulae in intersection 2: 0; common formulae: set()
# of common formulae in intersection 3: 0; common formulae: set()
```

Save split datasets to csv

Finally, after splitting the dataset into train/validation/test dataset splits, you can save them to disk for you to use later.

By saving these dataset splits into files, you can then later reproducibly use these same exact splits during your subsequent model training and comparison steps. Use the same datasets for all your models---that way, you can ensure a fair comparison.

Also, when you publish your results, you can include these dataset splits, so that others can use the exact datasets in their own studies.

```
In [11]: # saving these splits into csv files
PATH = os.getcwd()

train_path = os.path.join(PATH, '../data/cp_train.csv')
val_path = os.path.join(PATH, '../data/cp_val.csv')
test_path = os.path.join(PATH, '../data/cp_test.csv')

df_train.to_csv(train_path, index=False)
df_val.to_csv(val_path, index=False)
df_test.to_csv(test_path, index=False)
```

Remember, keep the test dataset locked away and forget about it until you have finalized your model! **Never look at the test dataset!!**

Data Featurization

Here, we will show some simple examples of featurizing materials composition data using so-called "composition-based feature vectors", or CBFVs. This methods represents a single chemical formula as one vector based on its constituent atoms' chemical properties (refer to the paper for more information and references).

Note that the steps shown in this notebook are intended to demonstrate the best practices associated with featurizing materials data, using *one* way of featurizing materials composition data as an example. Depending on your input data and your particular modeling needs, the data featurization method and procedure you use may be different than the example shown here.

```
In [1]: import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format='retina'

from collections import OrderedDict

# Set a random seed to ensure reproducibility across runs
RNG_SEED = 42
np.random.seed(RNG_SEED)
```

Loading data

We will start with the dataset splits that we saved from the last notebook.

```
In [2]: PATH = os.getcwd()
train_path = os.path.join(PATH, '../data/cp_train.csv')
val_path = os.path.join(PATH, '../data/cp_val.csv')
test_path = os.path.join(PATH, '../data/cp_test.csv')

df_train = pd.read_csv(train_path)
df_val = pd.read_csv(val_path)
df_test = pd.read_csv(test_path)

print(f'df_train DataFrame shape: {df_train.shape}')
print(f'df_val DataFrame shape: {df_val.shape}')
print(f'df_test DataFrame shape: {df_test.shape}')

df_train DataFrame shape: (3214, 3)
df_val DataFrame shape: (980, 3)
df_test DataFrame shape: (370, 3)
```

Sub-sampling your data (optional)

If your dataset is too large, you can subsample it to be a smaller size. This is useful for prototyping and for making quick sanity tests of new models / parameters.

Just be aware that you do not introduce any bias into your data through the sampling.

```
In [3]: # Sub-sample the data. Set the random_state to make the sampling reproducible
        # every time.
        df_train_sampled = df_train.sample(n=2000, random_state=RNG_SEED)
        df_val_sampled = df_val.sample(n=200, random_state=RNG_SEED)
        df_test_sampled = df_test.sample(n=200, random_state=RNG_SEED)

        print(f'df_train_sampled DataFrame shape: {df_train_sampled.shape}')
        print(f'df_val_sampled DataFrame shape: {df_val_sampled.shape}')
        print(f'df_test_sampled DataFrame shape: {df_test_sampled.shape}')

df_train_sampled DataFrame shape: (2000, 3)
df_val_sampled DataFrame shape: (200, 3)
df_test_sampled DataFrame shape: (200, 3)
```

Generate features using the CBFV package

To featurize the chemical compositions from a chemical formula (e.g. "Al2O3") into a composition-based feature vector (CBFV), we use the open-source [CBFV package \(https://github.com/kaaiian/CBFV\)](https://github.com/kaaiian/CBFV).

We have downloaded and saved a local copy of the package into this repository for your convenience. For the most updated version, refer to the GitHub repository linked above.

```
In [4]: # Import the package and the generate_features function
        from CBFV.cbfv.composition import generate_features
```

The `generate_features` function from the CBFV package expects an input DataFrame containing at least the columns `['formula', 'target']`. You may also have extra feature columns (e.g., temperature or pressure, other measurement conditions, etc.).

In our dataset, `Cp` represents the target variable, and `T` is the measurement condition. Since the `generate_features` function expects the target variable column to be named `target`, we have to rename the `Cp` column.

3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [5]: print('DataFrame column names before renaming:')
print(df_train.columns)
print(df_val.columns)
print(df_test.columns)

rename_dict = {'Cp': 'target'}
df_train = df_train.rename(columns=rename_dict)
df_val = df_val.rename(columns=rename_dict)
df_test = df_test.rename(columns=rename_dict)

df_train_sampled = df_train_sampled.rename(columns=rename_dict)
df_val_sampled = df_val_sampled.rename(columns=rename_dict)
df_test_sampled = df_test_sampled.rename(columns=rename_dict)

print('\nDataFrame column names after renaming:')
print(df_train.columns)
print(df_val.columns)
print(df_test.columns)
```

DataFrame column names before renaming:
Index(['formula', 'T', 'Cp'], dtype='object')
Index(['formula', 'T', 'Cp'], dtype='object')
Index(['formula', 'T', 'Cp'], dtype='object')

DataFrame column names after renaming:
Index(['formula', 'T', 'target'], dtype='object')
Index(['formula', 'T', 'target'], dtype='object')
Index(['formula', 'T', 'target'], dtype='object')

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

Now we can use the `generate_features` function to generate the CBFVs from the input data.

Note that we have specified several keyword arguments in our call to `generate_features` :

- `elem_prop='oliynyk'`
- `drop_duplicates=False`
- `extend_features=True`
- `sum_feat=True`

A short explanation for the choice of keyword arguments is below:

- The `elem_prop` parameter specifies which CBFV featurization scheme to use (there are several). For this tutorial, we have chosen to use the `oliynyk` CBFV featurization scheme.
- The `drop_duplicates` parameter specifies whether to drop duplicate formulae during featurization. In our case, we want to preserve duplicate formulae in our data (`True`), since we have multiple heat capacity measurements (performed at different temperatures) for the same compound.
- The `extend_features` parameter specifies whether to include extended features (features that are not part of [`'formula'`, `'target'`]) in the featurized data. In our case, this is our measurement temperature, and we want to include this information (`True`), since this is pertinent information for the heat capacity prediction.
- The `sum_feat` parameter specifies whether to calculate the sum features when generating the CBFVs for the chemical formulae. We do in our case (`True`).

For more information about the `generate_features` function and the CBFV featurization scheme, refer to the GitHub repository and the accompanying paper to this notebook.

3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [6]: X_train_unscaled, y_train, formulae_train, skipped_train = generate_features(df_train_sampled, elem_prop='oliynyk', drop_duplicates=False, extend_features=True, sum_feat=True)
X_val_unscaled, y_val, formulae_val, skipped_val = generate_features(df_val_sampled, elem_prop='oliynyk', drop_duplicates=False, extend_features=True, sum_feat=True)
X_test_unscaled, y_test, formulae_test, skipped_test = generate_features(df_test_sampled, elem_prop='oliynyk', drop_duplicates=False, extend_features=True, sum_feat=True)
```

```
Processing Input Data: 100%|██████████  
██████| 2000/2000 [00:00<00:00, 27852.38it/s]  
Assigning Features...: 0%|  
| 0/2000 [00:00<?, ?it/s]
```

Featurizing Compositions...

```
Assigning Features...: 100%|██████████|  
██████| 2000/2000 [00:00<00:00, 22532.15it/s]  
Processing Input Data: 100%|██████████|  
██████| 200/200 [00:00<00:00, 25067.56it/s]  
Assigning Features...: 100%|██████████|  
██████| 200/200 [00:00<00:00, 22281.68it/s]  
Processing Input Data: 0%|  
| 0/200 [00:00<?, ?it/s]
```

```
Creating Pandas Objects...
Featurizing Compositions...
Creating Pandas Objects...
```

```
Processing Input Data: 100%|██████████|  
██████| 200/200 [00:00<00:00, 22169.21it/s]  
Assigning Features...: 100%|██████████|  
██████| 200/200 [00:00<00:00, 20000.50it/s]
```

Featurizing Compositions...
Creating Pandas Objects...

To see what a featurized X matrix looks like, `.head()` will show us some rows:

```
In [7]: X_train_unscaled.head()
```

Out[7]:

	sum_Atomic_Number	sum_Atomic_Weight	sum_Period	sum_group	sum_families	sum_Metal
0	32.0	65.116040	8.0	30.0	15.0	1.0
1	28.0	53.491200	9.0	36.0	43.0	0.0
2	46.0	98.887792	14.0	72.0	36.0	3.0
3	20.0	41.988171	5.0	18.0	9.0	1.0
4	82.0	207.200000	6.0	14.0	5.0	1.0

5 rows × 177 columns

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [8]: X_train_unscaled.shape
```

```
Out[8]: (2000, 177)
```

Note the `sum` features in the CBFV, which we have included by using `sum_feat=True` in the call to `generate_features`.

Also note the temperature column `T` at the end of this featurized data.

What we have done above is featurize the input data. In the featurized data, each row contains a unique CBFV that describes a given chemical composition.

Data scaling & normalization

For numerical input data, scaling and normalization of the features often improves the model performance. Scaling can partially correct the discrepancy between the orders of magnitudes of the features (e.g., some numerical features being much larger or smaller than others). This typically improves the model learning performance, and in turn, improves the model performance.

We will scale then normalize our input data using scikit-learn's built-in `StandardScaler` class and `normalize` function.

Note, in addition to `StandardScaler`, other scalers such as `RobustScaler` and `MinMaxScaler` are also available in scikit-learn. Consult the documentation for the details and when to use them.

```
In [9]: from sklearn.preprocessing import StandardScaler
        from sklearn.preprocessing import normalize
```

Scaling the data

First, we instantiate the scaler object.

In a `StandardScaler` object:

- During the `fit` process, the statistics of the input data (mean and standard deviation) are computed.
- Then, during the `transform` process, the mean and standard deviation values calculated above are used to scale the data to having zero-mean and unit variance.

Therefore, for the first time usage of the scaler, we call the `.fit_transform()` method to fit the scaler to the input data, and then to transform the same data. For subsequent uses, since we have already computed the statistics, we only call the `.transform()` method to scale data.

Note: you should *only* `.fit()` the scaler using the training dataset statistics, and then use these same statistics from the training dataset to `.transform()` the other datasets (validation and train).

```
In [10]: scaler = StandardScaler()

X_train = scaler.fit_transform(X_train_unscaled)
X_val = scaler.transform(X_val_unscaled)
X_test = scaler.transform(X_test_unscaled)
```

Normalizing the scaled data

We repeat a similar process for normalizing the data. Here, there is no need to first fit the normalizer, since the normalizer scales the rows of the input data to unit norm independently of other rows.

The normalizer is different to a Scaler in that the normalizer acts row-wise, whereas a Scaler acts column-wise on the input data.

```
In [11]: X_train = normalize(X_train)
X_val = normalize(X_val)
X_test = normalize(X_test)
```

Modeling using "classical" machine learning models

Here we implement some classical ML models from `sklearn` :

- Ridge regression
- Support vector machine
- Linear support vector machine
- Random forest
- Extra trees
- Adaptive boosting
- Gradient boosting
- k-nearest neighbors
- Dummy (if you can't beat this, something is wrong.)

Note: the Dummy model types from `sklearn` act as a good sanity check for your ML studies. If your models do not perform significantly better than the equivalent Dummy models, then you should know that something has gone wrong in your model implementation.

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [12]: from time import time

from sklearn.dummy import DummyRegressor

from sklearn.linear_model import Ridge

from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import RandomForestRegressor

from sklearn.neighbors import KNeighborsRegressor

from sklearn.svm import SVR
from sklearn.svm import LinearSVR

from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
```

In addition, we define some helper functions.

3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [13]: def instantiate_model(model_name):
    model = model_name()
    return model

def fit_model(model, X_train, y_train):
    ti = time()
    model = instantiate_model(model)
    model.fit(X_train, y_train)
    fit_time = time() - ti
    return model, fit_time

def evaluate_model(model, X, y_act):
    y_pred = model.predict(X)
    r2 = r2_score(y_act, y_pred)
    mae = mean_absolute_error(y_act, y_pred)
    rmse_val = mean_squared_error(y_act, y_pred, squared=False)
    return r2, mae, rmse_val

def fit_evaluate_model(model, model_name, X_train, y_train, X_val, y_act_val):
    model, fit_time = fit_model(model, X_train, y_train)
    r2_train, mae_train, rmse_train = evaluate_model(model, X_train, y_train)
    r2_val, mae_val, rmse_val = evaluate_model(model, X_val, y_act_val)
    result_dict = {
        'model_name': model_name,
        'model_name_pretty': type(model).__name__,
        'model_params': model.get_params(),
        'fit_time': fit_time,
        'r2_train': r2_train,
        'mae_train': mae_train,
        'rmse_train': rmse_train,
        'r2_val': r2_val,
        'mae_val': mae_val,
        'rmse_val': rmse_val}
    return model, result_dict

def append_result_df(df, result_dict):
    df_result_appended = df.append(result_dict, ignore_index=True)
    return df_result_appended

def append_model_dict(dic, model_name, model):
    dic[model_name] = model
    return dic
```

Build an empty DataFrame to store model results:

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [14]: df_classics = pd.DataFrame(columns=['model_name',
                                             'model_name_pretty',
                                             'model_params',
                                             'fit_time',
                                             'r2_train',
                                             'mae_train',
                                             'rmse_train',
                                             'r2_val',
                                             'mae_val',
                                             'rmse_val'])

df_classics
```

```
Out[14]:
```

model_name	model_name_pretty	model_params	fit_time	r2_train	mae_train	rmse_train	r2_val
------------	-------------------	--------------	----------	----------	-----------	------------	--------

Define the models

Here, we instantiate several classical machine learning models for use. For demonstration purposes, we instantiate the models with their default model parameters.

Some of the models listed above can perform either regression or classification tasks. Because our ML task is a regression task (prediction of the continuous-valued target, heat capacity), we choose the regression variant of these models.

Note: the `DummyRegressor()` instance acts as a good sanity check for your ML studies. If your models do not perform significantly better than the `DummyRegressor()`, then you know something has gone awry.

```
In [15]: # Build a dictionary of model names
classic_model_names = OrderedDict({
    'dumr': DummyRegressor,
    'rr': Ridge,
    'abr': AdaBoostRegressor,
    'gbr': GradientBoostingRegressor,
    'rfr': RandomForestRegressor,
    'etr': ExtraTreesRegressor,
    'svr': SVR,
    'lsvr': LinearSVR,
    'knn': KNeighborsRegressor,
})
```

Instantiate and fit the models

Now, we can fit the ML models.

We will loop through each of the models listed above. For each of the models, we will:

- instantiate the model (with default parameters)
- fit the model using the training data
- use the fitted model to generate predictions from the validation data
- evaluate the performance of the model using the predictions
- store the results in a DataFrame for analysis

Note: this may take several minutes, depending on your hardware/software environment, dataset size and featurization scheme (CBFV).

```
In [16]: # Instantiate a dictionary to store the model objects
         classic_models = OrderedDict()

         # Keep track of elapsed time
         ti = time()

         # Loop through each model type, fit and predict, and evaluate and store results
         for model_name, model in classic_model_names.items():
             print(f'Now fitting and evaluating model {model_name}: {model.__name__}')
             model, result_dict = fit_evaluate_model(model, model_name, X_train, y_train, X_val, y_val)
             df_classics = append_result_df(df_classics, result_dict)
             classic_models = append_model_dict(classic_models, model_name, model)

         dt = time() - ti
         print(f'Finished fitting {len(classic_models)} models, total time: {dt:0.2f} s')
```

```
Now fitting and evaluating model dumr: DummyRegressor
Now fitting and evaluating model rr: Ridge
Now fitting and evaluating model abr: AdaBoostRegressor
Now fitting and evaluating model gbr: GradientBoostingRegressor
Now fitting and evaluating model rfr: RandomForestRegressor
Now fitting and evaluating model etr: ExtraTreesRegressor
Now fitting and evaluating model svr: SVR
Now fitting and evaluating model lsvr: LinearSVR
Now fitting and evaluating model knr: KNeighborsRegressor
Finished fitting 9 models, total time: 21.38 s
```

Now, we can look at the results.

You will notice, that some of the models (such as RandomForestRegressor, ExtraTreesRegressor and GradientBoostingRegressor) have completely memorized the training data, as evidenced by the very high `r2_train` scores of ~1.0.

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [17]: # Sort in order of increasing validation r2 score
df_classics = df_classics.sort_values('r2_val', ignore_index=True)
df_classics
```

Out[17]:

	model_name	model_name_pretty	model_params	fit_time	r2_train	mae_train	rmse_
0	dumr	DummyRegressor	{'constant': None, 'quantile': None, 'strategy...	0.000000	0.000000	47.451805	60.60
1	lsvr	LinearSVR	{'C': 1.0, 'dual': True, 'epsilon': 0.0, 'fit_...	0.008975	0.763989	16.961421	29.44
2	svr	SVR	{'C': 1.0, 'cache_size': 200, 'coef0': 0.0, 'd...	0.653226	0.763306	16.726278	29.48
3	knr	KNeighborsRegressor	{'algorithm': 'auto', 'leaf_size': 30, 'metric...	0.046875	0.981784	3.902671	8.17
4	abr	AdaBoostRegressor	{'base_estimator': None, 'learning_rate': 1.0, ...	1.740368	0.922253	14.174608	16.89
5	rr	Ridge	{'alpha': 1.0, 'copy_X': True, 'fit_intercept'...	0.027902	0.875762	14.425002	21.36
6	rfr	RandomForestRegressor	{'bootstrap': True, 'ccp_alpha': 0.0, 'criteri...	8.986937	0.998316	1.216542	2.48
7	etr	ExtraTreesRegressor	{'bootstrap': False, 'ccp_alpha': 0.0, 'criter...	4.108041	0.999995	0.010897	0.13
8	gbr	GradientBoostingRegressor	{'alpha': 0.9, 'ccp_alpha': 0.0, 'criterion': ...	4.803637	0.985051	5.360182	7.40

You can now also access the full details of the models by inspecting the `classic_models` dictionary that we populated.

3.2 PUBLICATION 1: BESTPRACTICES – SI

In [18]: `classic_models`

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```

Out[18]: OrderedDict([('dumr',
                        DummyRegressor(constant=None, quantile=None, strategy='mean')),
                        ('rr',
                         Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=Non
e,
                                normalize=False, random_state=None, solver='auto', tol=0.
001)),
                        ('abr',
                         AdaBoostRegressor(base_estimator=None, learning_rate=1.0, loss
='linear',
                                           n_estimators=50, random_state=None)),
                        ('gbr',
                         GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion
='friedman_mse',
                                                    init=None, learning_rate=0.1, loss='l
s', max_depth=3,
                                                    max_features=None, max_leaf_nodes=Non
e,
                                                    min_impurity_decrease=0.0, min_impuri
ty_split=None,
                                                    min_samples_leaf=1, min_samples_split
=2,
                                                    min_weight_fraction_leaf=0.0, n_estim
ators=100,
                                                    n_iter_no_change=None, presort='depre
cated',
                                                    random_state=None, subsample=1.0, tol
=0.0001,
                                                    validation_fraction=0.1, verbose=0, w
arm_start=False)),
                        ('rfr',
                         RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion
='mse',
                                              max_depth=None, max_features='auto', max_
leaf_nodes=None,
                                              max_samples=None, min_impurity_decrease=
0.0,
                                              min_impurity_split=None, min_samples_leaf
=1,
                                              min_samples_split=2, min_weight_fraction_
leaf=0.0,
                                              n_estimators=100, n_jobs=None, oob_score=
False,
                                              random_state=None, verbose=0, warm_start=
False)),
                        ('etr',
                         ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0, criterion
='mse',
                                              max_depth=None, max_features='auto', max_le
af_nodes=None,
                                              max_samples=None, min_impurity_decrease=0.
0,
                                              min_impurity_split=None, min_samples_leaf=
1,
                                              min_samples_split=2, min_weight_fraction_le
af=0.0,
                                              n_estimators=100, n_jobs=None, oob_score=Fa

```

3.2 PUBLICATION 1: BESTPRACTICES – SI

```
lse,
                                random_state=None, verbose=0, warm_start=False),
lse)),
    ('svr',
     SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=0.001,
         kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)),
    ('lsvr',
     LinearSVR(C=1.0, dual=True, epsilon=0.0, fit_intercept=True,
               intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=1000,
               random_state=None, tol=0.001, verbose=0)),
    ('knnr',
     KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='min_kowski',
                        metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                        weights='uniform'))))
```

Evaluating model performance on validation dataset

Now comes the time to evaluate the trained models on the validation set.

Remember, we use the same validation set to evaluate all models. This ensures a fair comparison.

In addition, we plot the predicted vs. actual plots using the predictions made by each trained model on the same validation set.

```
In [19]: def plot_pred_act(act, pred, model, reg_line=True, label=''):
        xy_max = np.max([np.max(act), np.max(pred)])

        plot = plt.figure(figsize=(6,6))
        plt.plot(act, pred, 'o', ms=9, mec='k', mfc='silver', alpha=0.4)
        plt.plot([0, xy_max], [0, xy_max], 'k--', label='ideal')
        if reg_line:
            polyfit = np.polyfit(act, pred, deg=1)
            reg_ys = np.poly1d(polyfit)(np.unique(act))
            plt.plot(np.unique(act), reg_ys, alpha=0.8, label='linear fit')
        plt.axis('scaled')
        plt.xlabel(f'Actual {label}')
        plt.ylabel(f'Predicted {label}')
        plt.title(f'{type(model).__name__}, r2: {r2_score(act, pred):0.4f}')
        plt.legend(loc='upper left')

        return plot
```

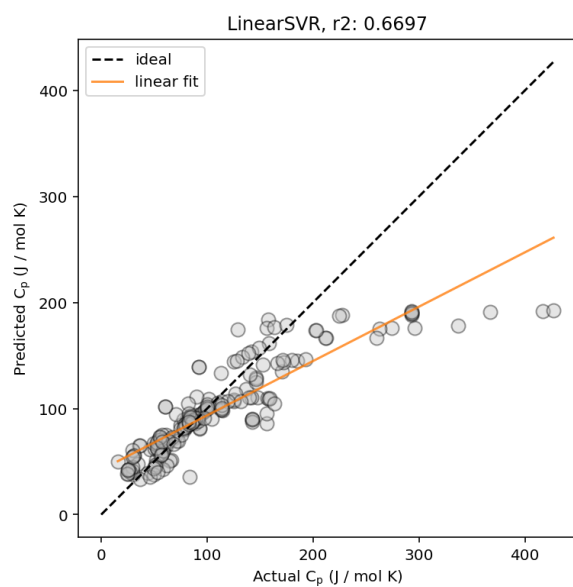
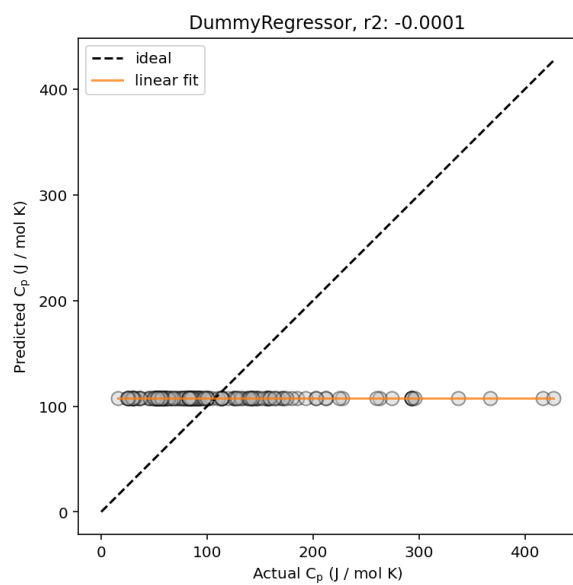
3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [20]: for row in range(df_classics.shape[0]):
          model_name = df_classics.iloc[row]['model_name']

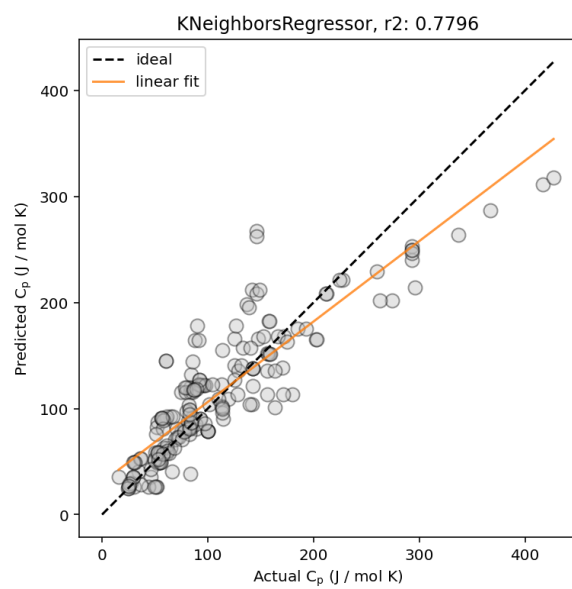
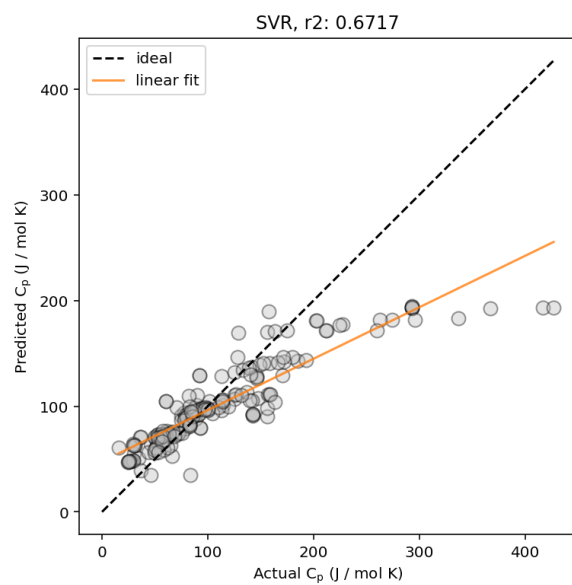
          model = classic_models[model_name]
          y_act_val = y_val
          y_pred_val = model.predict(X_val)

          plot = plot_pred_act(y_act_val, y_pred_val, model, reg_line=True, label=
                              '$\mathrm{C}_{\mathrm{p}}$ (J / mol K)')
```

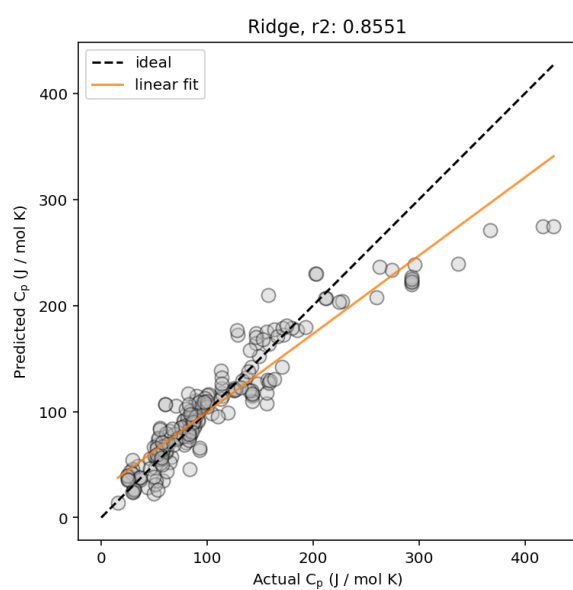
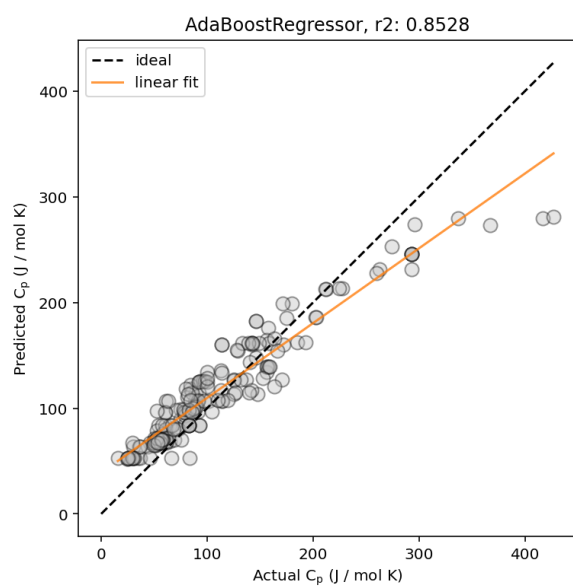

3.2 PUBLICATION 1: BESTPRACTICES – SI



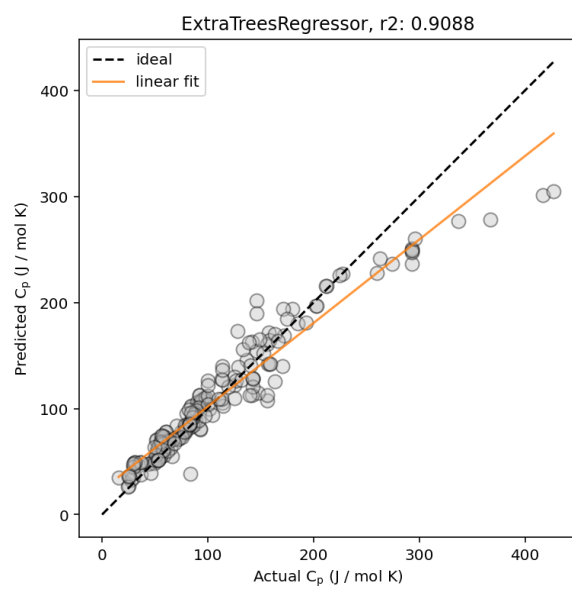
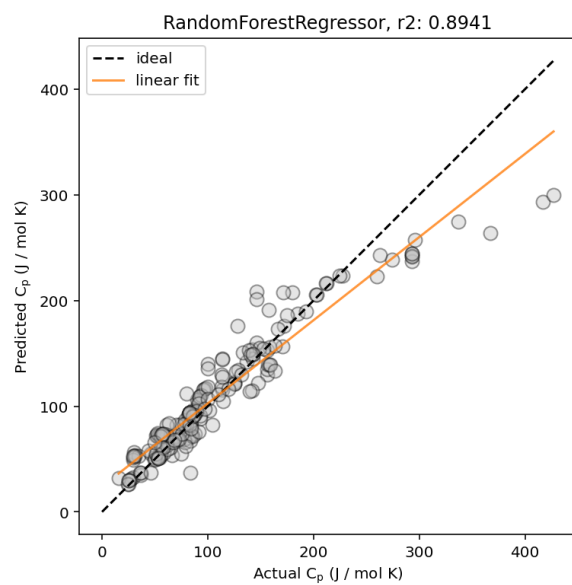
3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE



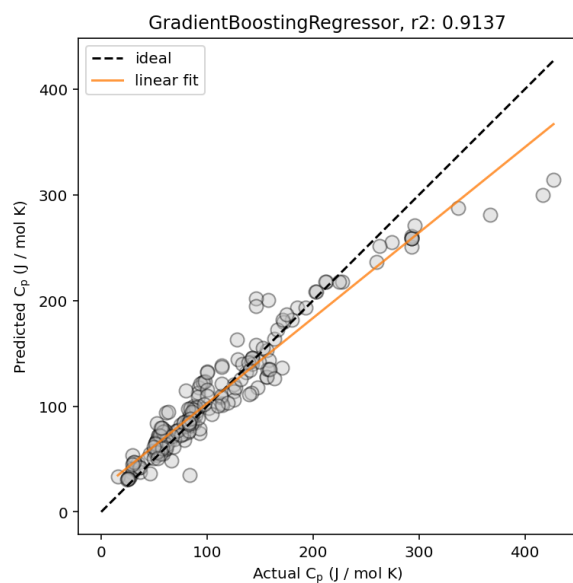
3.2 PUBLICATION 1: BESTPRACTICES – SI



3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE



3.2 PUBLICATION 1: BESTPRACTICES – SI



Re-training the best-performing model on combined train + validation dataset

After you have finalized your model, you can re-train your model (using the same hyperparameters) again on the combined train + validation datasets, and finally, evaluate your model on the held-out test dataset.

By training on the combined train + validation dataset after you have finished tuning your model, you give it more training data, which should lead to an increase in the model performance.

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [21]: # Find the best-performing model that we have tested
best_row = df_classics.iloc[-1, :].copy()

# Get the model type and model parameters
model_name = best_row['model_name']
model_params = best_row['model_params']

# Instantiate the model again using the parameters
model = classic_model_names[model_name](**model_params)
print(model)

GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                           init=None, learning_rate=0.1, loss='ls', max_depth=
3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0, warm_start=False)

In [22]: # Concatenate the train and validation datasets together
X_train_new = np.concatenate((X_train, X_val), axis=0)
y_train_new = pd.concat((y_train, y_val), axis=0)

print(X_train_new.shape)

(2200, 177)
```

Finally, we can fit the model on the combined train + validation dataset.

```
In [23]: ti = time()

model.fit(X_train_new, y_train_new)

dt = time() - ti
print(f'Finished fitting best model, total time: {dt:0.2f} s')

Finished fitting best model, total time: 5.44 s
```

Testing the re-trained model on the test dataset

After re-fitting the best model on the train+validation dataset, you can finally test it on the test dataset.

Remember: you should only do this *once*!

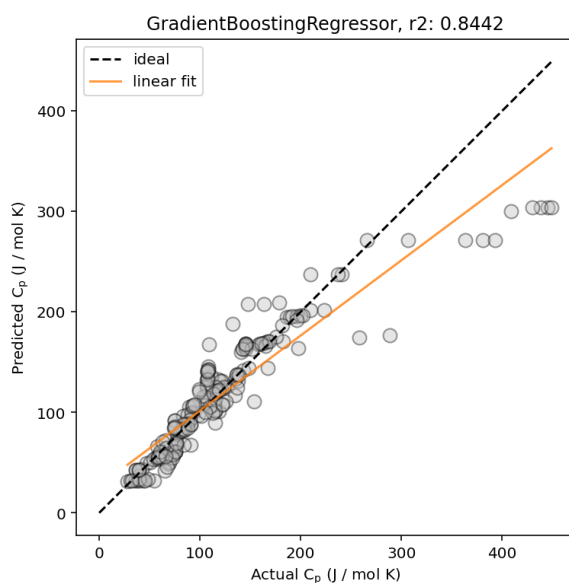
3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [24]: y_act_test = y_test
y_pred_test = model.predict(X_test)

r2, mae, rmse = evaluate_model(model, X_test, y_test)
print(f'r2: {r2:0.4f}')
print(f'mae: {mae:0.4f}')
print(f'rmse: {rmse:0.4f}')

plot = plot_pred_act(y_act_test, y_pred_test, model, reg_line=True, label='$\mathrm{C}_p$ (J / mol K)')
```

r2: 0.8442
mae: 17.1236
rmse: 31.0274



We see that our model achieves decent performance on the held-out test dataset.

Effect of train/validation/test dataset split

Using different train/validation/test splits can dramatically affect your model performance, even for classical ML models.

Here, we provide a little demonstration.

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [25]: X_train_unscaled, y_train, formulae_train, skipped_train = generate_features(df_train, elem_prop='oliynyk', drop_duplicates=False, extend_features=True, sum_feat=True)
X_val_unscaled, y_val, formulae_val, skipped_val = generate_features(df_val, elem_prop='oliynyk', drop_duplicates=False, extend_features=True, sum_feat=True)
X_test_unscaled, y_test, formulae_test, skipped_test = generate_features(df_test, elem_prop='oliynyk', drop_duplicates=False, extend_features=True, sum_feat=True)
```

```
Processing Input Data: 100%|██████████|  
██████| 3214/3214 [00:00<00:00, 28511.32it/s]  
Assigning Features...: 100%|██████████|  
██████| 3214/3214 [00:00<00:00, 22850.93it/s]
```

Featurizing Compositions...
Creating Pandas Objects...

```
Processing Input Data: 100%|██████████|  
██████████| 980/980 [00:00<00:00, 28877.87it/s]  
Assigning Features...: 100%|██████████|  
██████████| 980/980 [00:00<00:00, 29644.50it/s]  
Processing Input Data: 100%|██████████|  
██████████| 370/370 [00:00<00:00, 23672.78it/s]  
Assigning Features...: 100%|██████████|  
██████████| 370/370 [00:00<00:00, 18795.10it/s]
```

```
Featurizing Compositions...
Creating Pandas Objects...
Featurizing Compositions...
Creating Pandas Objects...
```

```
In [26]: X_train_original = X_train_unscaled.copy()
X_val = X_val_unscaled.copy()
X_test = X_test_unscaled.copy()

y_train_original = y_train.copy()
```

We sample the training data using 10 random seeds, by using the `DataFrame.sample()` method with seeds ranging from 0 to 9. We then fit 10 models, each on one of the random splits, and evaluate their performance on the same validation dataset.

3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [27]: splits = range(10)
df_splits = pd.DataFrame(columns=['split',
                                  'r2_train',
                                  'mae_train',
                                  'rmse_train',
                                  'r2_val',
                                  'mae_val',
                                  'rmse_val'])

for split in splits:
    print(f'Fitting and evaluating random split {split}')
    X_train = X_train_original.sample(frac=0.7, random_state=split)
    y_train = y_train_original[X_train.index]

    scaler = StandardScaler()
    X_train = normalize(scaler.fit_transform(X_train))
    X_val = normalize(scaler.transform(X_val_unscaled))
    X_test = normalize(scaler.transform(X_test_unscaled))

    model = AdaBoostRegressor()
    model.fit(X_train, y_train)
    y_act_val = y_val
    y_pred_val = model.predict(X_val)

    r2_train, mae_train, rmse_train = evaluate_model(model, X_train, y_train)
    r2_val, mae_val, rmse_val = evaluate_model(model, X_val, y_val)
    result_dict = {
        'split': split,
        'r2_train': r2_train,
        'mae_train': mae_train,
        'rmse_train': rmse_train,
        'r2_val': r2_val,
        'mae_val': mae_val,
        'rmse_val': rmse_val}

    df_splits = append_result_df(df_splits, result_dict)
```

```
Fitting and evaluating random split 0
Fitting and evaluating random split 1
Fitting and evaluating random split 2
Fitting and evaluating random split 3
Fitting and evaluating random split 4
Fitting and evaluating random split 5
Fitting and evaluating random split 6
Fitting and evaluating random split 7
Fitting and evaluating random split 8
Fitting and evaluating random split 9
```

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [28]: df_splits['split'] = df_splits['split'].astype(int)
df_splits
```

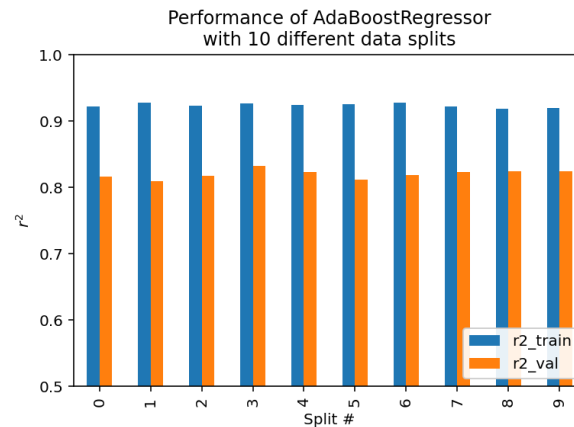
Out[28]:

	split	r2_train	mae_train	rmse_train	r2_val	mae_val	rmse_val
0	0	0.921647	14.237594	16.979393	0.815858	23.865538	34.461629
1	1	0.927656	13.501629	16.265329	0.808852	22.688912	35.111086
2	2	0.922515	14.448165	17.079006	0.817551	23.744004	34.302802
3	3	0.926511	13.651451	16.537507	0.831742	20.315320	32.941742
4	4	0.924336	13.983030	16.703454	0.822637	22.490512	33.821357
5	5	0.925542	13.714406	16.610536	0.811329	23.408679	34.882859
6	6	0.927083	13.872571	16.567055	0.818648	23.621438	34.199530
7	7	0.921361	14.275176	16.893382	0.823097	23.931573	33.777476
8	8	0.918063	14.508259	17.281824	0.824275	23.186429	33.664755
9	9	0.919256	14.204361	16.984472	0.824302	23.008472	33.662190

We then plot the train and validation r^2 scores for each of the 10 models.

Note the high variability in the $r2_val$ score. In contrast, the variability in the $r2_train$ score is comparatively lower.

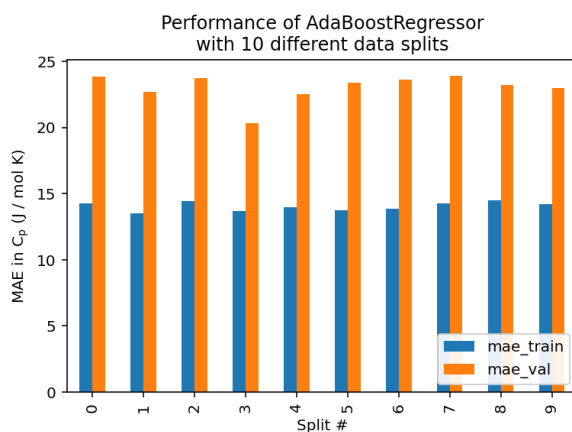
```
In [29]: df_splits.plot('split', ['r2_train', 'r2_val'], kind='bar')
plt.title(f'Performance of {type(model).__name__}\nwith {len(splits)} different data splits')
plt.ylim((0.5, 1.0))
plt.ylabel('$r^2$')
plt.xlabel('Split #')
plt.legend(loc='lower right', framealpha=0.9)
plt.show()
```



3.2 PUBLICATION 1: BESTPRACTICES – SI

This effect is even more pronounced when we plot the mean absolute error (MAE).

```
In [30]: df_splits.plot('split', ['mae_train', 'mae_val'], kind='bar')
plt.title(f'Performance of {type(model).__name__}\nwith {len(splits)} different
t data splits')
plt.ylabel('MAE in  $C_p$  (J / mol K)')
plt.xlabel('Split #')
plt.legend(loc='lower right', framealpha=0.9)
plt.show()
```



Therefore, typically the average value of all the scores are reported, as this gives a much more accurate estimate of how well the model actually performs.

```
In [31]: avg_r2_val = df_splits['r2_val'].mean()
avg_mae_val = df_splits['mae_val'].mean()

print(f'Average validation r2: {avg_r2_val:0.4f}')
print(f'Average validation MAE: {avg_mae_val:0.4f}')
```

Average validation r2: 0.8198
Average validation MAE: 23.0261

Modeling using neural network / deep learning-based models

In this notebook, we will cover how to implement a simple neural network for the modeling of heat capacity.

We will load, prepare featurize, and scale/normalize the input datasets the same way as we did in the previous notebook. For more information about the individual steps, you can consult that notebook.

```
In [1]: import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format='retina'

from collections import OrderedDict

from CBFV.cbfv.composition import generate_features

from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize

from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

import torch
from torch import nn
from torch.utils.data import Dataset, DataLoader

import torch.optim as optim

# Set a random seed to ensure reproducibility across runs
RNG_SEED = 42
np.random.seed(RNG_SEED)
torch.manual_seed(RNG_SEED)
```

```
Out[1]: <torch._C.Generator at 0x24e8007e510>
```

Featurizing and scaling data

Nothing new here---same steps as we've done in the previous notebook.

3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [2]: PATH = os.getcwd()
train_path = os.path.join(PATH, '../data/cp_train.csv')
val_path = os.path.join(PATH, '../data/cp_val.csv')
test_path = os.path.join(PATH, '../data/cp_test.csv')

df_train = pd.read_csv(train_path)
df_val = pd.read_csv(val_path)
df_test = pd.read_csv(test_path)

print(f'df_train DataFrame shape: {df_train.shape}')
print(f'df_val DataFrame shape: {df_val.shape}')
print(f'df_test DataFrame shape: {df_test.shape}')

df_train DataFrame shape: (3214, 3)
df_val DataFrame shape: (980, 3)
df_test DataFrame shape: (370, 3)
```

Here we do not sub-sample the datasets into smaller datasets like we did in the previous notebook. Typically, the more data you have for neural networks, the better the networks will be able to train, and the better they will perform (as long as they are well-conditioned).

Additionally, the performance of PyTorch is very good for modern computers, especially if you have a modern CUDA-capable graphics processing unit (GPU) such as an Nvidia GPU to accelerate the computations. Our dataset is small enough to fit into almost all modern computers or CUDA-capable GPUs.

80

```
In [3]: rename_dict = {'Cp': 'target'}  
df_train = df_train.rename(columns=rename_dict)  
df_val = df_val.rename(columns=rename_dict)  
df_test = df_test.rename(columns=rename_dict)  
  
X_train_unscaled, y_train, formulae_train, skipped_train = generate_features(df_train, elem_prop='oliynyk', drop_duplicates=False, extend_features=True, sum_feat=True)  
X_val_unscaled, y_val, formulae_val, skipped_val = generate_features(df_val, elem_prop='oliynyk', drop_duplicates=False, extend_features=True, sum_feat=True)  
X_test_unscaled, y_test, formulae_test, skipped_test = generate_features(df_test, elem_prop='oliynyk', drop_duplicates=False, extend_features=True, sum_feat=True)
```

```
Processing Input Data: 100%|███████████  
██████| 3214/3214 [00:00<00:00, 28074.71it/s]  
Assigning Features...: 0%|  
| 0/3214 [00:00<?, ?it/s]  
  
Featurizing Compositions...  
  
Assigning Features...: 100%|███████████  
██████| 3214/3214 [00:00<00:00, 22408.81it/s]  
  
Creating Pandas Objects...  
  
Processing Input Data: 100%|███████████  
██████| 980/980 [00:00<00:00, 28050.95it/s]  
Assigning Features...: 100%|███████████  
██████| 980/980 [00:00<00:00, 23381.48it/s]  
Processing Input Data: 100%|███████████  
██████| 370/370 [00:00<00:00, 28536.88it/s]  
  
Featurizing Compositions...  
Creating Pandas Objects...  
  
Assigning Features...: 100%|███████████  
██████| 370/370 [00:00<00:00, 21791.04it/s]  
  
Featurizing Compositions...  
Creating Pandas Objects...
```

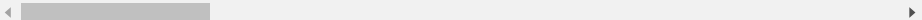
3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [4]: X_train_unscaled.head()
```

```
Out[4]:
```

	sum_Atomic_Number	sum_Atomic_Weight	sum_Period	sum_group	sum_families	sum_Metal
0	34.0	69.6202	10.0	74.0	33.0	0.0
1	34.0	69.6202	10.0	74.0	33.0	0.0
2	34.0	69.6202	10.0	74.0	33.0	0.0
3	34.0	69.6202	10.0	74.0	33.0	0.0
4	34.0	69.6202	10.0	74.0	33.0	0.0

5 rows × 177 columns



```
In [5]: scaler = StandardScaler()

X_train = scaler.fit_transform(X_train_unscaled)
X_val = scaler.transform(X_val_unscaled)
X_test = scaler.transform(X_test_unscaled)

X_train = normalize(X_train)
X_val = normalize(X_val)
X_test = normalize(X_test)
```

Building a neural network

This is where you get to be the architect, and design your own neural network!

For sake of clarity (and to ensure that this tutorial runs on all the potatoes of this world), we will define a simple dense fully-connected neural network (which we will call `DenseNet`) as an example.

The input layer of `DenseNet` accepts input data in the dimension of each row of the input data, which is equal to the number of features in our CBFV featurization scheme. In our particular example, when featurized using the `oliynyk` featurizer, the input dimension is 177 (it is the second dimension when you view `X_train.shape`).

The output layer dimension of `DenseNet` is 1, because we want to predict one value (heat capacity).

In addition, `DenseNet` can have one or more "hidden layers" that are attached between the input and output layers. These can be any arbitrary dimensions > 1 you want to choose.

Defining the network in PyTorch

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [6]: class DenseNet(nn.Module):
        """
        This implements a dynamically-built dense fully-connected neural network
        with Leaky ReLU activation and optional dropout.

        Parameters
        -----
        input_dims: int
            Number of input features (required).
        hidden_dims: list of ints
            Number of hidden features, where each integer represents the number of
            hidden features in each subsequent hidden linear layer (optional,
            default=[64, 32]).
        output_dims: int
            Number of output features (optional, default=1).
        dropout: float
            the dropout value (optional, default=0.0).
        """
        def __init__(self,
                      input_dims,
                      hidden_dims=[64, 32],
                      output_dims=1,
                      dropout=0.0):
            super().__init__()

            self.input_dims = input_dims
            self.hidden_dims = hidden_dims
            self.output_dims = output_dims

            self.dropout = dropout

            # Build a sub-block of linear networks
            def fc_block(in_dim, out_dim, *args, **kwargs):
                return nn.Sequential(
                    nn.Linear(in_dim, out_dim, *args, **kwargs),
                    nn.Dropout(p=self.dropout),
                    nn.LeakyReLU()
                )

            # Build overall network architecture
            self.network = nn.ModuleList([
                nn.Sequential(
                    nn.Linear(input_dims, self.hidden_dims[0]),
                    nn.Dropout(p=self.dropout),
                    nn.LeakyReLU()
                )
            ])

            hidden_layer_sizes = zip(self.hidden_dims[:-1], self.hidden_dims[1:])
            self.network.extend([
                fc_block(in_dim, out_dim) for in_dim, out_dim
                in hidden_layer_sizes
            ])

            self.network.extend([
                nn.Linear(hidden_dims[-1], output_dims)]
```


3.2 PUBLICATION 1: BESTPRACTICES – SI

```
)

def forward(self, x):
    """
    Forward pass of the DenseNet model.

    Parameters
    -----
    x: torch.Tensor
        A representation of the chemical compounds in the shape
        (n_compounds, n_feats).

    Returns
    -----
    y: torch.Tensor
        The element property prediction with the shape 1.
    """
    for i, subnet in enumerate(self.network):
        x = subnet(x)

    y = x

    return y
```

Specifying the compute device for calculations

Before we run the neural network, we can first check if your machine has a CUDA-capable device. CUDA is a specific set of application instructions (application programming interfaces, APIs) that PyTorch can use to accelerate some of the calculations performed in neural networks.

Generally, a relatively recent GPU from Nvidia will support CUDA capabilities, and can be used to accelerate neural network computations in PyTorch.

In case you do not have a CUDA-capable device, PyTorch will fall back to using the CPU. Depending on the complexity of your model, training and predicting using a CPU can take significantly longer than using a CUDA-capable GPU.

Consult the [PyTorch](https://pytorch.org/docs/stable/torch.html) (<https://pytorch.org/docs/stable/torch.html>) and [CUDA](https://docs.nvidia.com/cuda/) (<https://docs.nvidia.com/cuda/>) documentation for more information.

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [7]: CUDA_available = torch.cuda.is_available()
print(f'CUDA is available: {CUDA_available}')

if CUDA_available:
    compute_device = torch.device('cuda')
else:
    compute_device = torch.device('cpu')

print(f'Compute device for PyTorch: {compute_device}')
```

CUDA is available: True
Compute device for PyTorch: cuda

Defining the data loader and dataset structure

Here we define a dataloader class specific for loading CBFV-type datasets.

We also define the CBFV dataset class that tells PyTorch how our dataset is structured, and how to grab individual data samples from our dataset.

3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [8]: class CBFVDataLoader():
        """
        Parameters
        -----
        train_data: np.ndarray or pd.DataFrame or pd.Series
            name of csv file containing cif and properties
        val_data: np.ndarray or pd.DataFrame or pd.Series
            name of csv file containing cif and properties
        test_data: np.ndarray or pd.DataFrame or pd.Series
            name of csv file containing cif and properties
        batch_size: float, optional (default=64)
            Step size for the Gaussian filter
        random_state: int, optional (default=42)
            Random seed for sampling the dataset. Only used if validation data is
            not given.
        shuffle: bool, optional (default=True)
            Whether to shuffle the datasets or not
        """
        def __init__(self, train_data, val_data, test_data,
                     batch_size=64, num_workers=1, random_state=42,
                     shuffle=True, pin_memory=True):

            self.train_data = train_data
            self.val_data = val_data
            self.test_data = test_data

            self.batch_size = batch_size
            self.num_workers = num_workers
            self.pin_memory = pin_memory

            self.shuffle = shuffle
            self.random_state = random_state

        def get_data_loaders(self, batch_size=1):
            """
            Input the dataset, get train test split
            """
            train_dataset = CBFVDataset(self.train_data)
            val_dataset = CBFVDataset(self.val_data)
            test_dataset = CBFVDataset(self.test_data)

            train_loader = DataLoader(train_dataset,
                                     batch_size=self.batch_size,
                                     pin_memory=self.pin_memory,
                                     shuffle=self.shuffle)

            val_loader = DataLoader(val_dataset,
                                   batch_size=self.batch_size,
                                   pin_memory=self.pin_memory,
                                   shuffle=self.shuffle)

            test_loader = DataLoader(test_dataset,
                                    batch_size=self.batch_size,
                                    pin_memory=self.pin_memory,
                                    shuffle=False)
```

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
        return train_loader, val_loader, test_loader

class CBFVDataset(Dataset):
    """
    Get X and y from CBFV-based dataset.
    """
    def __init__(self, dataset):
        self.data = dataset

        self.X = np.array(self.data[0])
        self.y = np.array(self.data[1])
        self.shape = [(self.X.shape), (self.y.shape)]

    def __str__(self):
        string = f'CBFVDataset with X.shape {self.X.shape}'
        return string

    def __len__(self):
        return self.X.shape[0]

    def __getitem__(self, idx):
        X = self.X[[idx], :]
        y = self.y[idx]

        X = torch.as_tensor(X)
        y = torch.as_tensor(np.array(y))

        return (X, y)
```

Here we choose a batch size for loading data, and initialize the DataLoader for loading the featurized input data.

We also get the data loaders corresponding to the train, validation, and test datasets.

```
In [9]: train_data = (X_train, y_train)
        val_data = (X_val, y_val)
        test_data = (X_test, y_test)

        # Instantiate the DataLoader
        batch_size = 128
        data_loaders = CBFVDataLoader(train_data, val_data, test_data, batch_size=batch_size)
        train_loader, val_loader, test_loader = data_loaders.get_data_loaders()
```

Instantiating a DenseNet model

Now, we can instantiate... an instance of the DenseNet model.

3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [10]: # Get input dimension size from the dataset
example_data = train_loader.dataset.data[0]
input_dims = example_data.shape[-1]

# Instantiate the model
model = DenseNet(input_dims, hidden_dims=[16], dropout=0.0)
print(model)

DenseNet(
  (network): ModuleList(
    (0): Sequential(
      (0): Linear(in_features=177, out_features=16, bias=True)
      (1): Dropout(p=0.0, inplace=False)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (1): Linear(in_features=16, out_features=1, bias=True)
  )
)
```

Defining the loss criterion & optimizer

Here, we see the model and its individual layers and components printed nicely.

We then instantiate and initialize the loss criterion and optimizer.

Note, there are many choices of loss criteria and optimizers that are provided by PyTorch, each with their benefits and limitations, and a myriad of parameters. Consult the PyTorch documentation for further details.

```
In [11]: # Initialize the loss criterion
criterion = nn.L1Loss()
print('Loss criterion: ')
print(criterion)

# Initialize the optimizer
optim_lr = 1e-2
optimizer = optim.Adam(model.parameters(), lr=optim_lr)
print('\nOptimizer: ')
print(optimizer)

Loss criterion:
L1Loss()

Optimizer:
Adam (
  Parameter Group 0
    amsgrad: False
    betas: (0.9, 0.999)
    eps: 1e-08
    lr: 0.01
    weight_decay: 0
)
```

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

Moving the model to the compute device

Then, we can move the model and loss criterion computation to the compute device.

If you have a GPU, this will transfer and attach the required resources to the GPU. If you have a CPU, then everything will remain on the CPU.

```
In [12]: # Move the model and criterion to the compute device
         model = model.to(compute_device)
         criterion = criterion.to(compute_device)
```

Defining some additional helper functions

We define some scalar functions and helper functions to evaluate and visualize model results.

3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [13]: class Scaler():
    def __init__(self, data):
        self.data = torch.as_tensor(data)
        self.mean = torch.mean(self.data)
        self.std = torch.std(self.data)

    def scale(self, data):
        data = torch.as_tensor(data)
        data_scaled = (data - self.mean) / self.std
        return data_scaled

    def unscale(self, data_scaled):
        data_scaled = torch.as_tensor(data_scaled)
        data = data_scaled * self.std + self.mean
        return data

    def state_dict(self):
        return {'mean': self.mean,
                'std': self.std}

    def load_state_dict(self, state_dict):
        self.mean = state_dict['mean']
        self.std = state_dict['std']

class MeanLogNormScaler():
    def __init__(self, data):
        self.data = torch.as_tensor(data)
        self.logdata = torch.log(self.data)
        self.mean = torch.mean(self.logdata)
        self.std = torch.std(self.logdata)

    def scale(self, data):
        data = torch.as_tensor(data)
        data_scaled = (torch.log(data) - self.mean) / self.std
        return data_scaled

    def unscale(self, data_scaled):
        data_scaled = torch.as_tensor(data_scaled) * self.std + self.mean
        data = torch.exp(data_scaled)
        return data

    def state_dict(self):
        return {'mean': self.mean,
                'std': self.std}

    def load_state_dict(self, state_dict):
        self.mean = state_dict['mean']
        self.std = state_dict['std']
```

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [14]: def predict(model, data_loader):
    target_list = []
    pred_list = []

    model.eval()
    with torch.no_grad():
        for i, data_output in enumerate(data_loader):
            X, y_act = data_output
            X = X.to(compute_device,
                      dtype=data_type,
                      non_blocking=True)
            y_act = y_act.cpu().flatten().tolist()
            y_pred = model.forward(X).cpu().flatten().tolist()

            # Unscale target values
            y_pred = target_scaler.unscale(y_pred).tolist()

            targets = y_act
            predictions = y_pred
            target_list.extend(targets)
            pred_list.extend(predictions)
    model.train()

    return target_list, pred_list

def evaluate(target, pred):
    r2 = r2_score(target, pred)
    mae = mean_absolute_error(target, pred)
    rmse = mean_squared_error(target, pred, squared=False)
    output = (r2, mae, rmse)
    return output

def print_scores(scores, label=''):
    r2, mae, rmse = scores
    print(f'{label} r2: {r2:0.4f}')
    print(f'{label} mae: {mae:0.4f}')
    print(f'{label} rmse: {rmse:0.4f}')
    return scores

def plot_pred_act(act, pred, model, reg_line=True, label=''):
    xy_max = np.max([np.max(act), np.max(pred)])

    plot = plt.figure(figsize=(6,6))
    plt.plot(act, pred, 'o', ms=9, mec='k', mfc='silver', alpha=0.4)
    plt.plot([0, xy_max], [0, xy_max], 'k--', label='ideal')
    if reg_line:
        polyfit = np.polyfit(act, pred, deg=1)
        reg_ys = np.poly1d(polyfit)(np.unique(act))
        plt.plot(np.unique(act), reg_ys, alpha=0.8, label='linear fit')
    plt.axis('scaled')
    plt.xlabel(f'Actual {label}')
    plt.ylabel(f'Predicted {label}')
    plt.title(f'{type(model).__name__}, r2: {r2_score(act, pred):0.4f}')
```


3.2 PUBLICATION 1: BESTPRACTICES – SI

```
plt.legend(loc='upper left')  
  
return plot
```

We scale the target variables.

```
In [15]: y_train = [data[1].numpy().tolist() for data in train_loader]  
y_train = [item for sublist in y_train for item in sublist]  
  
y_train = train_loader.dataset.y  
target_scaler = MeanLogNormScaler(y_train)
```

Training the neural network

And finally, we train the neural network.

This is the training procedure for the neural network:

- for each epoch :
 - iterate through the train dataset using `train_loader` :
 - scale the target data
 - transfer input (`X`) and target (`y`) data to compute device
 - reset the optimizer's gradient to zero
 - compute the output of the model (forward pass)
 - calculate the loss of the model (between the predicted and true target values)
 - propagate the loss backwards through the model (backpropagation)
 - update the weights throughout the model
 - if `epoch == print_every` :
 - print the current epoch (to keep track of training progress)
 - if `epoch == plot_every` :
 - evaluate the model on the validation dataset using `val_loader`
 - plot predicted vs. actual value plots
 - print the train and val r^2 , MAE and RMSE scores of the model

Note: training this network may take up to tens of minutes, depending on your hardware configuration and whether or not you have a CUDA-capable device.

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [16]: data_type = torch.float
epochs = 500

print_every = 20
plot_every = 50

for epoch in range(epochs):
    if epoch % print_every == 0 or epoch == epochs - 1:
        print(f'epoch: {epoch}')
    if epoch % plot_every == 0:
        target_train, pred_train = predict(model, train_loader)
        train_scores = evaluate(target_train, pred_train)
        print_scores(train_scores, label='train')

        target_val, pred_val = predict(model, val_loader)
        val_scores = evaluate(target_val, pred_val)
        print_scores(val_scores, label='val')
        plot_pred_act(target_val, pred_val, model, label='${\mathrm{C}}_{\mathrm{p}}$ (J / mol K)')
        plt.show()

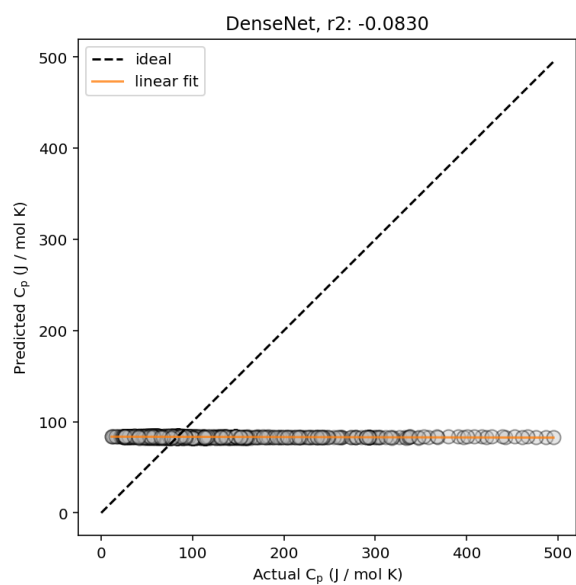
    for i, data_output in enumerate(train_loader):
        X, y = data_output
        y = target_scaler.scale(y)

        X = X.to(compute_device,
                  dtype=data_type,
                  non_blocking=True)
        y = y.to(compute_device,
                  dtype=data_type,
                  non_blocking=True)

        optimizer.zero_grad()
        output = model.forward(X).flatten()
        loss = criterion(output.view(-1), y.view(-1))
        loss.backward()
        optimizer.step()
```

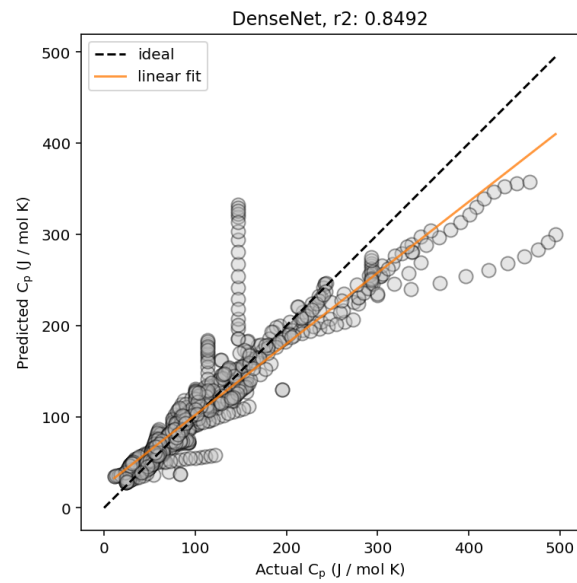
3.2 PUBLICATION 1: BESTPRACTICES – SI

epoch: 0
train r2: -0.1487
train mae: 46.4704
train rmse: 64.9909
val r2: -0.0830
val mae: 52.6671
val rmse: 83.5762

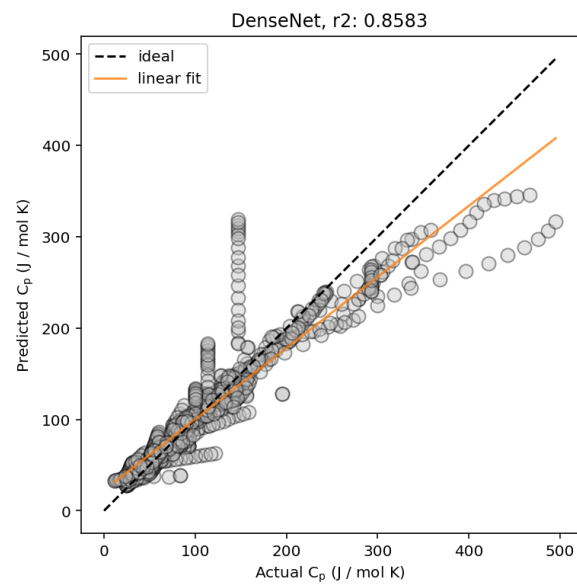


epoch: 20
epoch: 40
train r2: 0.9813
train mae: 4.3085
train rmse: 8.2817
val r2: 0.8492
val mae: 16.8082
val rmse: 31.1895

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

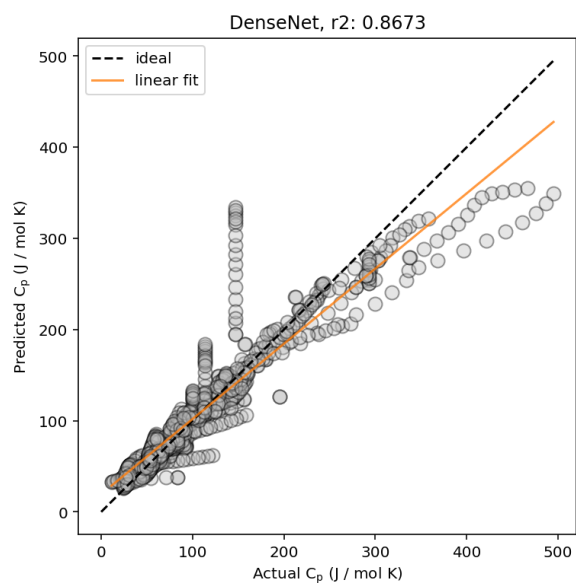


epoch: 60
epoch: 80
epoch: 100
train r2: 0.9833
train mae: 3.9003
train rmse: 7.8337
val r2: 0.8583
val mae: 16.2121
val rmse: 30.2252



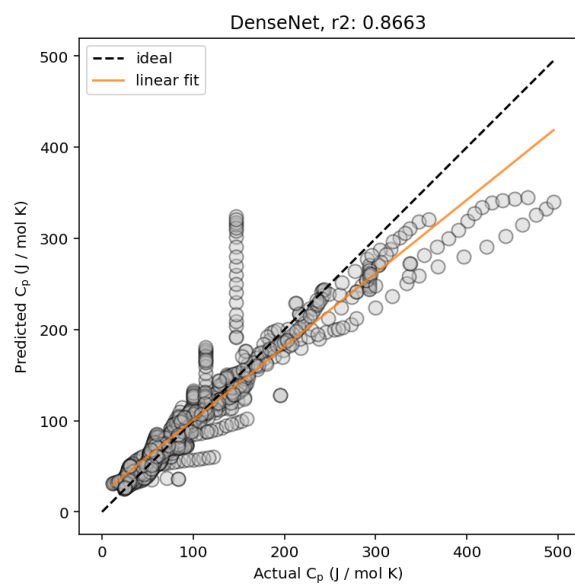
3.2 PUBLICATION 1: BESTPRACTICES – SI

epoch: 120
epoch: 140
train r2: 0.9840
train mae: 3.6218
train rmse: 7.6821
val r2: 0.8673
val mae: 15.5079
val rmse: 29.2527

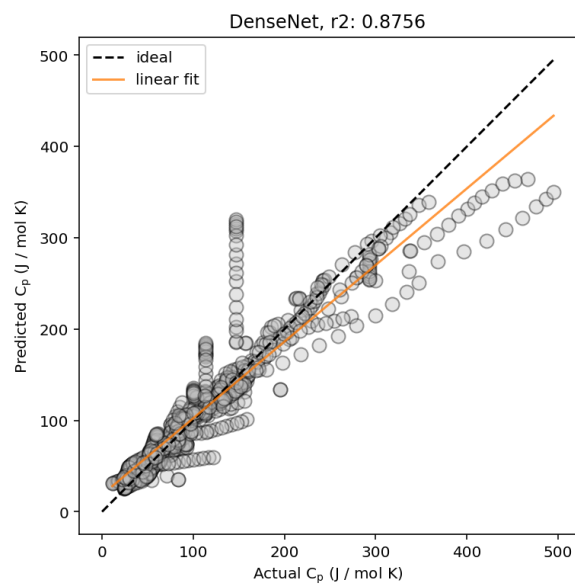


epoch: 160
epoch: 180
epoch: 200
train r2: 0.9866
train mae: 3.1012
train rmse: 7.0212
val r2: 0.8663
val mae: 15.8610
val rmse: 29.3698

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

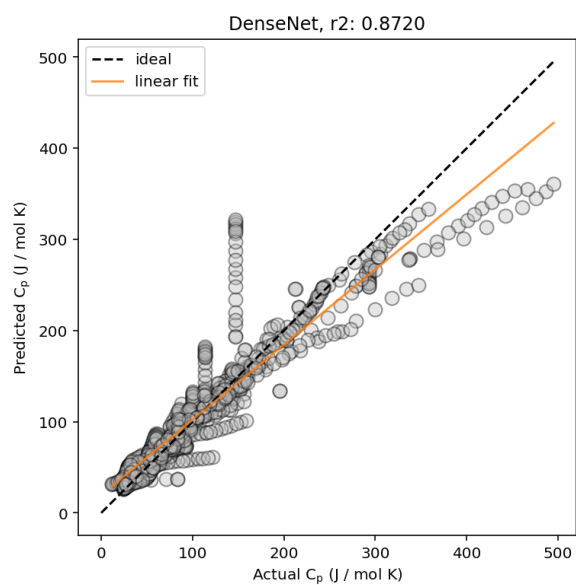


epoch: 220
epoch: 240
train r2: 0.9861
train mae: 3.4834
train rmse: 7.1619
val r2: 0.8756
val mae: 15.2939
val rmse: 28.3246



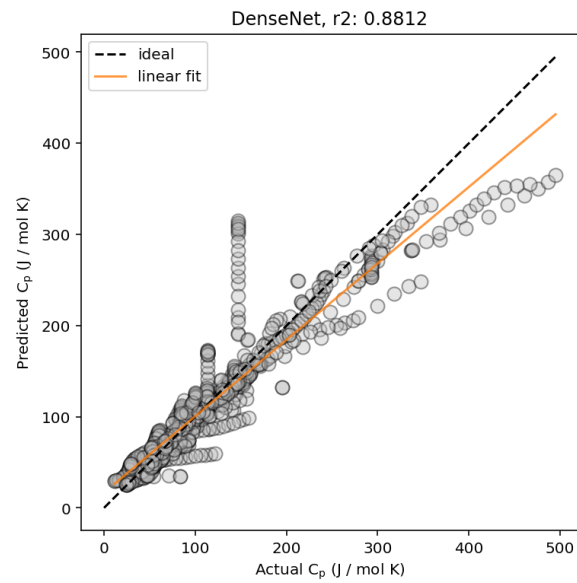
3.2 PUBLICATION 1: BESTPRACTICES – SI

epoch: 260
epoch: 280
epoch: 300
train r2: 0.9872
train mae: 3.0628
train rmse: 6.8711
val r2: 0.8720
val mae: 15.8923
val rmse: 28.7362

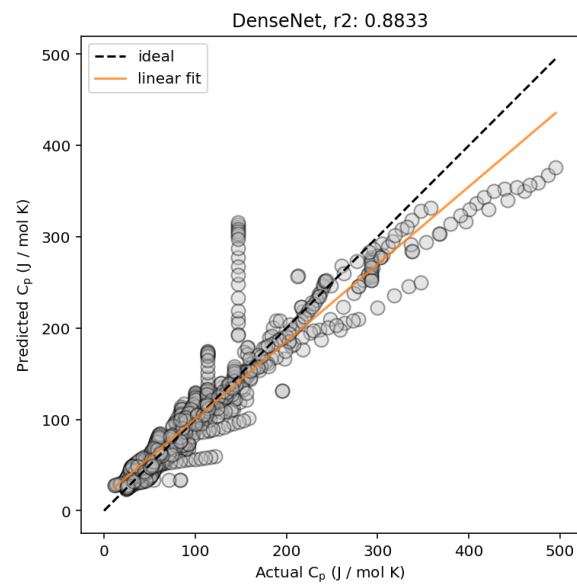


epoch: 320
epoch: 340
train r2: 0.9874
train mae: 3.0447
train rmse: 6.8130
val r2: 0.8812
val mae: 15.2294
val rmse: 27.6838

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

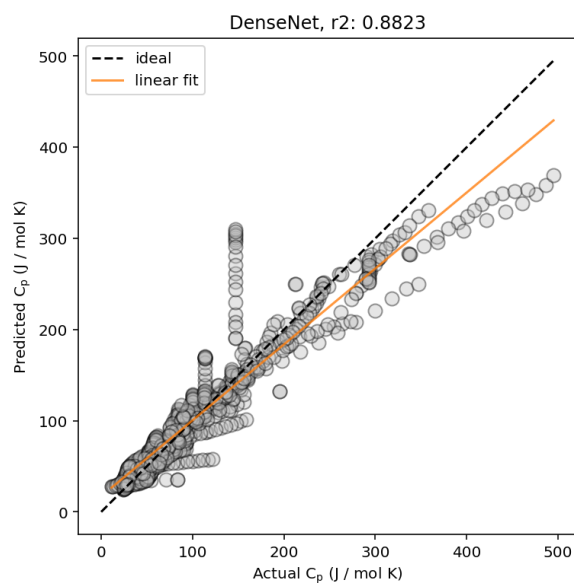


epoch: 360
epoch: 380
epoch: 400
train r2: 0.9872
train mae: 3.2069
train rmse: 6.8730
val r2: 0.8833
val mae: 15.1164
val rmse: 27.4352



3.2 PUBLICATION 1: BESTPRACTICES – SI

epoch: 420
epoch: 440
train r2: 0.9878
train mae: 2.9436
train rmse: 6.7003
val r2: 0.8823
val mae: 15.3402
val rmse: 27.5488



epoch: 460
epoch: 480
epoch: 499

Now, with our trained neural network, we can evaluate the performance of the model (at the end of the training phase) on the validation dataset.

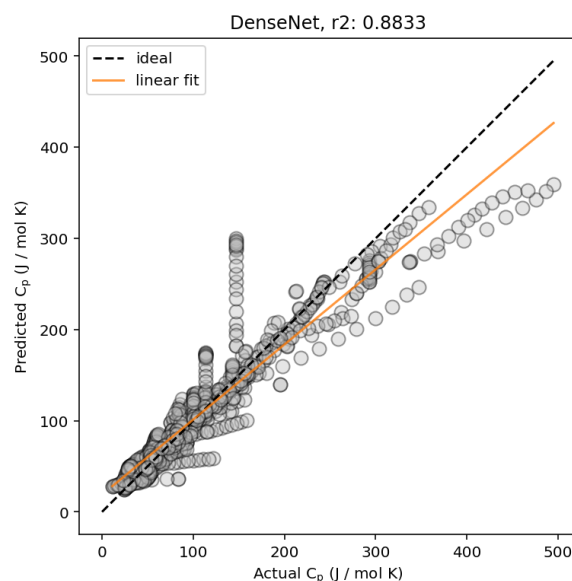
3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [17]: target_val, pred_val = predict(model, val_loader)
        scores = evaluate(target_val, pred_val)

        print_scores(scores, label='val')

        plot = plot_pred_act(target_val, pred_val, model, label='${\mathrm{C}}_p\mathrm{ }_{\mathrm{p}}$ (J / mol K)')

val r2: 0.8833
val mae: 15.3854
val rmse: 27.4286
```



Keeping track of training progress -- avoid overfitting

Note, you can keep track of the training progress by saving the train and validation metrics such as r^2 and MAE at every epoch. Then, you can plot so-called "loss curves" that show the loss of the model vs. epoch throughout the training process. This gives you additional insight into your model training process, and helps you diagnose issues such as overfitting, improper model/optimizer/loss parameters, and so on.

Once you start tracking these performance metrics during your training loop, you can also implement more advanced training techniques such as "early stopping". In early stopping, you observe the performance metrics (such as validation r^2 or MAE) over the training epochs, and you stop the training process if you observe that the metrics are not improving any more (meaning your model is fully trained), or if the metrics are increasing again after reaching a minimum (meaning your model is overfitting the training set).

Evaluating model performance on test dataset

And finally evaluate the performance on the test dataset. **Remember:** you should only do this *once*!

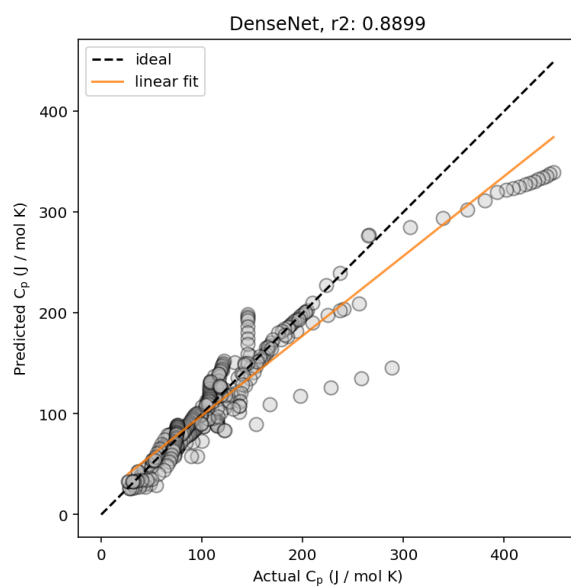
3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [18]: target_test, pred_test = predict(model, test_loader)
scores = evaluate(target_test, pred_test)

print_scores(scores, label='test')

plot = plot_pred_act(target_test, pred_test, model, label=' $\mathrm{C}_p$  (J / mol K)')

test r2: 0.8899
test mae: 14.2758
test rmse: 25.8626
```



Exporting PyTorch models

Now that we've got a (reasonably well-performing) model, we can export the weights and biases from the model to what is referred to as a "checkpoint" file.

The advantages of exporting your model to a checkpoint file are manifold. For example, when you want to re-use the model again later (to make further predictions, or even to continue training), you don't have to train the model from scratch again. For our current `DenseNet` model, this may not seem like a big deal, since it trains within minutes. But once you start moving on to larger and larger models, model training time can reach hours, days---even weeks!

Another advantage is that you can greatly enhance the reproducibility of your work. If you export your models, other researchers can then recreate your model architecture on their system, then load your weights into the model to get exactly the model you trained. This allows them to use your model as-is, and enables them to reproduce your work---an important step if they are to judge the merit of your work.

With that said, we will now use PyTorch's built-in methods to export (1) our `DenseNet` model, and (2) our `target_scaler` (we need to export our `target_scaler` object as well, because we need to use it to unscale the model predictions to get back the true prediction values).

Saving the model + target scaler

```
In [19]: save_dict = {'weights': model.state_dict(),
                    'scaler_state': target_scaler.state_dict()}
print(save_dict)

{'weights': OrderedDict([('network.0.0.weight', tensor([[ 0.7553,  0.9386,
0.3842, ..., 0.0225, 0.4568, 2.6430],
[ 0.2389,  0.2387,  0.1926, ..., -0.3318,  0.0269,  3.4188],
[-1.3899, -1.3630, -0.7993, ...,  1.0238,  0.6904, -3.7036],
...,
[-0.8147, -0.8653, -0.6919, ...,  1.1902,  1.1137, -2.3598],
[ 0.4363,  0.4713,  0.2978, ..., -1.2085, -0.8724,  1.3804],
[ 0.2154,  0.4221, -0.0204, ..., -0.4609, -0.0218,  2.9495]],
device='cuda:0')), ('network.0.0.bias', tensor([-2.3551e-01, -1.4669e-
01,  1.0728e-02,  1.2264e-01, -5.2010e-01,
-5.2636e-01, -1.8640e-01, -2.3231e-01, -1.4808e-01,  4.8181e-02,
 3.2689e-01, -8.2749e-02, -5.8574e-01, -1.0106e-01,  1.7105e-01,
-8.1883e-06], device='cuda:0')), ('network.1.weight', tensor([[ 0.227
2,  0.1508, -0.1535,  0.0698, -0.5781, -0.6211,  0.2401,  0.0881,
 0.2408, -0.1805, -0.2659,  0.2306, -0.3568, -0.1981,  0.1904,  0.14
82]],
device='cuda:0')), ('network.1.bias', tensor([-0.0122], device='cuda:
0')))), 'scaler_state': {'mean': tensor(4.5152, dtype=torch.float64), 'std':
tensor(0.5749, dtype=torch.float64)}}
```

3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [20]: pth_path = ('model_checkpoint.pth') # .pth is commonly used as the file extension
         torch.save(save_dict, pth_path)
```

Navigate to your notebooks directory. You should now find a file named 'model_checkpoint.pth'. Since the DenseNet model is small, the checkpoint file weighs in relatively lightly at 13KB. Bigger models will have more weights & biases, and will require more storage space for the checkpoint file.

Loading the model + target scaler

Of course, if you provide the facilities to **save** a model, you should also provide facilities to **load** them and to recreate your model back.

Thankfully, PyTorch makes this also easy.

```
In [21]: # First, clear the variables for model and target_scaler.
         # We want to start with a clean slate.
         model = None
         target_scaler = None
         del model
         del target_scaler
```

We start by recreating the DenseNet model and the target_scaler that we originally built. This model will be initialized with random weights & biases, which we will then overload (overwrite) afterwards with the values from the checkpoint file.

Make sure that you create the same model and target_scaler here as the ones you saved the checkpoint file from. Otherwise you will not be able to load the checkpoint file, or it will produce unexpected results.

3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [22]: # Instantiate the model.  
# The model will be randomly initialized, but we will overwrite  
# all weights and biases when we load the checkpoint.  
model = DenseNet(input_dims, hidden_dims=[16], dropout=0.0)  
model = model.to(compute_device)  
print(model)
```

```
# Instantiate the target_scaler.  
# We initialize this target_scaler with a vector of zeros,  
# but we will overwrite its internal parameters  
# when we load the checkpoint.  
target_scaler = MeanLogNormScaler(torch.zeros(42))  
  
DenseNet(  
  (network): ModuleList(  
    (0): Sequential(  
      (0): Linear(in_features=177, out_features=16, bias=True)  
      (1): Dropout(p=0.0, inplace=False)  
      (2): LeakyReLU(negative_slope=0.01)  
    )  
    (1): Linear(in_features=16, out_features=1, bias=True)  
  )  
)
```

```
In [23]: # Load the checkpoint and map it to the compute device  
pth_path = ('model_checkpoint.pth')  
checkpoint = torch.load(pth_path, map_location=compute_device)  
  
# Load the state dictionaries back into the model and target_scaler  
model.load_state_dict(checkpoint['weights'])  
target_scaler.load_state_dict(checkpoint['scaler_state'])
```

Checking the loaded model

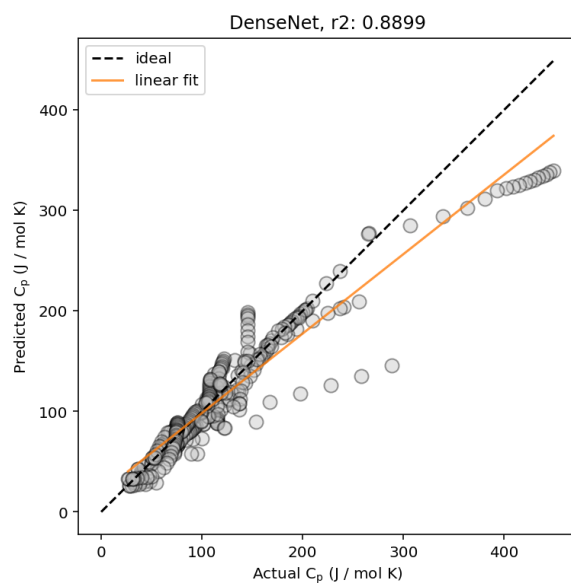
3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [24]: target_test, pred_test = predict(model, test_loader)
        scores = evaluate(target_test, pred_test)

        print_scores(scores, label='test')

        plot = plot_pred_act(target_test, pred_test, model, label=' $\mathrm{C}_p$  (J / mol K)')

test r2: 0.8899
test mae: 14.2758
test rmse: 25.8626
```



Hooray!

Visualizing results

Here, we will show some typical examples of visualizations that are used often to show results in ML studies in materials science.

We will use the open-source `ML_figures` [package \(https://github.com/kaaiian/ML_figures\)](https://github.com/kaaiian/ML_figures) and the example data provided by the package to generate these figures.

```
In [1]: import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format='retina'

# Import the ML_figures package and the figure-plotting functions
from ML_figures.figures import act_pred
from ML_figures.figures import residual, residual_hist
from ML_figures.figures import loss_curve
from ML_figures.figures import element_prevalence
```

Predicted vs. actual value plots

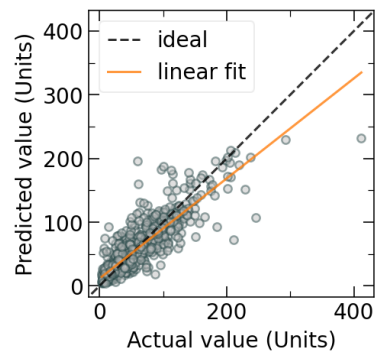
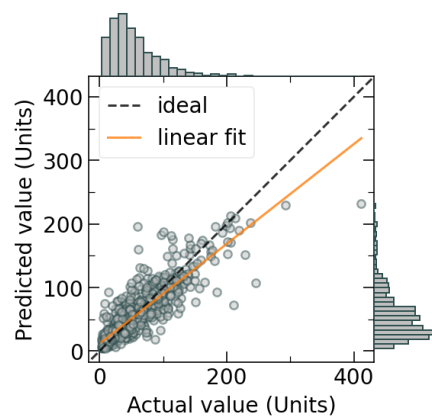
These plots, you have already seen before in the previous notebooks.

3.2 PUBLICATION 1: BESTPRACTICES – SI

```
In [2]: # Read in example act vs. pred data
df_act_pred = pd.read_csv('ML_figures/example_data/act_pred.csv')
y_act, y_pred = df_act_pred.iloc[:, 1], df_act_pred.iloc[:, 2]

act_pred(y_act, y_pred,
         reg_line=True,
         save_dir='ML_figures/example_figures')

act_pred(y_act, y_pred,
         name='example_no_hist',
         x_hist=False, y_hist=False,
         reg_line=True,
         save_dir='ML_figures/example_figures')
```



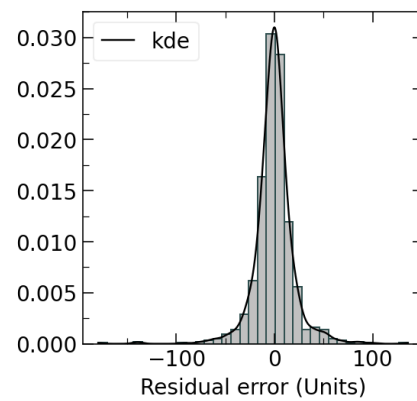
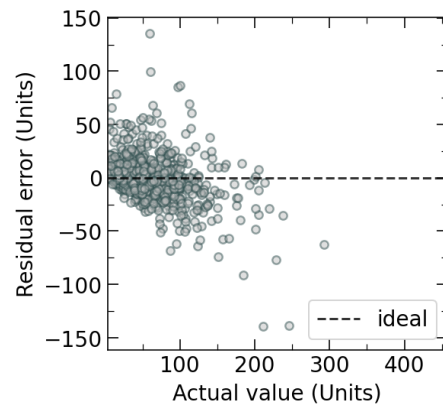
Residual error plots

Residual error plots show how far your model's predictions deviate from the actual values. They are using the same data used in the predicted vs. actual plots; however, instead of plotting predicted vs. actual values, residual error plots plot (predicted - actual) vs. actual values.

This lets you visually analyze your model's prediction error on a straight horizontal line.

Alternatively, you can plot the residual errors on a histogram, and optionally with a kernel density estimation (kde).

```
In [3]: residual(y_act, y_pred,  
               save_dir='ML_figures/example_figures')  
  
residual_hist(y_act, y_pred,  
              save_dir='ML_figures/example_figures')
```



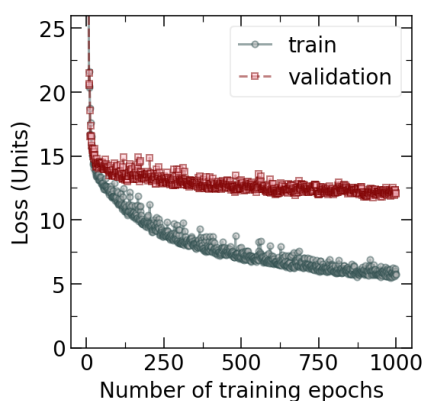
Loss curves

Loss curves show the loss of a neural network model vs. epoch throughout the training process. It is typically evaluated using the training and validation dataset at the end of each epoch (or every n epochs, where n is a small number, if evaluating every epoch takes too many resources).

Typically, loss curves plot the model performance (such as r^2 score) or loss (such as MAE) against epoch.

```
In [4]: # Read in Loss curve data
df_lc = pd.read_csv('ML_figures/example_data/training_progress.csv')
epoch = df_lc['epoch']
train_err, val_err = df_lc['mae_train'], df_lc['mae_val']

loss_curve(epoch, train_err, val_err,
            save_dir='ML_figures/example_figures')
```



Visualizing elemental prevalence

Depending on your dataset, what you are studying, and how the compounds/constituent elements of the compounds in the dataset are distributed, it may be useful to visualize the elemental prevalence in your dataset.

These figures let you visualize the relative amount of certain elements vs. other elements present in your dataset, and can help you in identifying dataset biases, imbalanced datasets, or other issues.

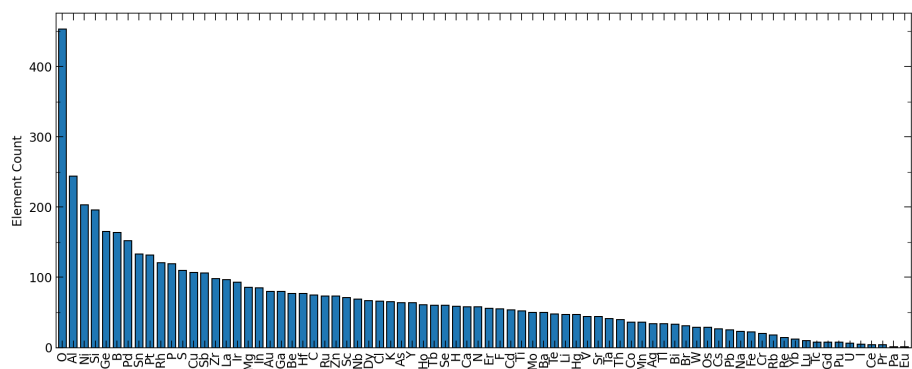
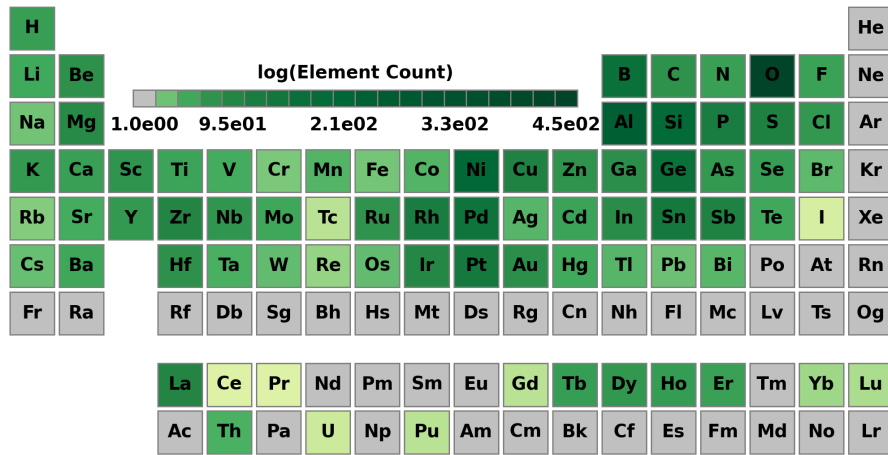
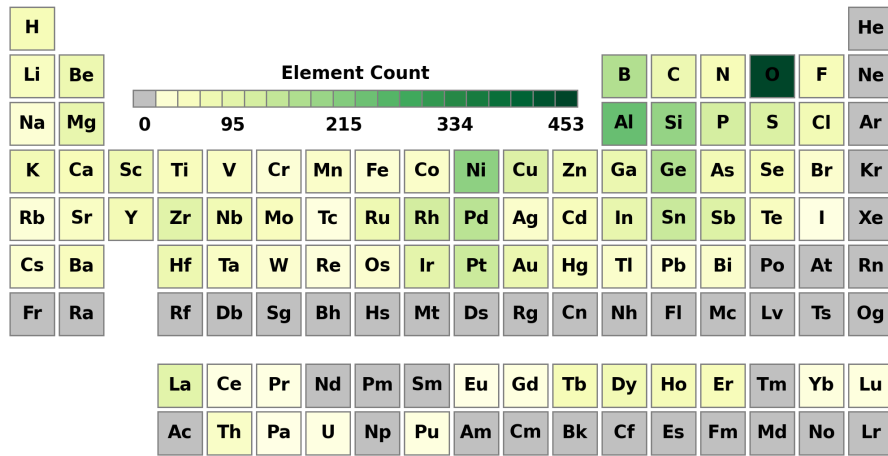
3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE

```
In [5]: # Visualize element prevalence
formula = df_act_pred.iloc[:, 0]

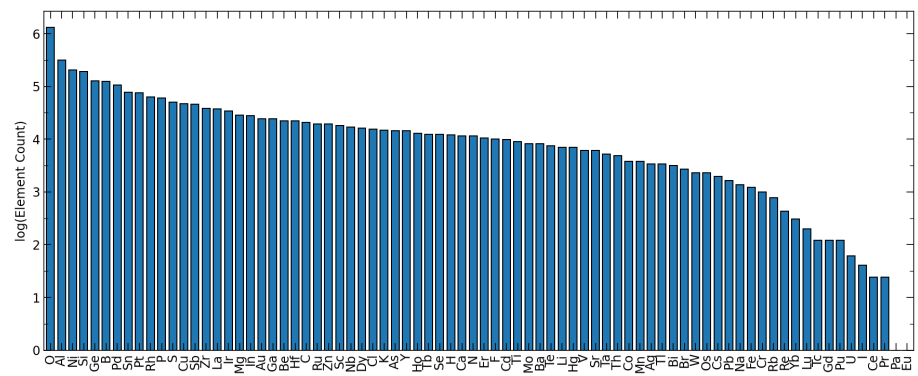
element_prevalence(formula,
                    save_dir='ML_figures/example_figures',
                    log_scale=False)
element_prevalence(formula,
                    save_dir='ML_figures/example_figures',
                    name='example_log',
                    log_scale=True)

plt.rcParams.update({'font.size': 12})
element_prevalence(formula,
                    save_dir='ML_figures/example_figures',
                    ptable_fig=False,
                    log_scale=False)
element_prevalence(formula,
                    save_dir='ML_figures/example_figures',
                    name='example_log',
                    ptable_fig=False,
                    log_scale=True)
```

3.2 PUBLICATION 1: BESTPRACTICES – SI



3 BEST PRACTICES FOR MACHINE LEARNING IN MATERIALS SCIENCE



4 Novel attention-based learning of materials properties

4.1 Lack of adequate composition featurization techniques

Classical ML methods such as linear regression, random forest, and support vector machines have been successfully used for the regression and classification of many material properties [19, 52, 69, 98–101]. In order to train these classical ML methods, the input chemical formulae are first transformed into numerical features that can then be used by the models. This process is also referred to as “featurization”. Typically, featurization of chemical formulae is achieved through the use of a composition-based feature vector (CBFV), which uses weighted descriptive statistics (such as the mean, variance, range, and more) of the properties of constituent elements in each chemical compound to uniquely represent the compound [4, 37, 52, 68, 102–104]. Examples of common CBFV feature sets for element properties are shown in Table 4.1 and include Oliynyk, Magpie, Jarvis, Atom2Vec, ElemNet, and mat2vec.

Table 4.1: Examples of common composition-based feature vector (CBFV) feature sets, including their source and requirements on domain knowledge and hand-engineering.

Feature set	Source	Domain knowledge	Hand-engineering	Ref.
Oliynyk	Physical	High	High	[69]
Magpie	Physical	High	High	[52]
Jarvis	Physical	High	High	[105]
Atom2Vec	Computational	Medium	None to low	[106]
ElemNet	Computational	Low	None to low	[107]
mat2vec	Computational	Medium	None to low	[70]

Here, a distinction is made between physically-derived CBFVs (with features based on measurable element properties) and computationally-derived CBFVs (with features

obtained from computational or deep learning models). It is also important to note that the use of these CBFV feature sets relies on (1) different amounts of domain knowledge in materials science to understand the feature origins and their usage, and (2) varying amounts of effort in hand-engineering the features to suit the ML task at hand. Further analysis about the different CBFV feature sets and their utility for the machine learning of materials properties can be found in [4, 104].

The CBFV-based featurization of chemical compositions has been shown to work well for many ML tasks in materials science. However, there is one main assumption inherent to this approach that ultimately limits its potential for being applied to all materials properties. This assumption is that the chemical compounds should be featurized based on the stoichiometric weighting of their element vectors. In the CBFV featurization approach (Figure 4.1), the vectors representing the individual elements in the compound are typically scaled by the element's fractional prevalence (*i.e.*, the stoichiometry in the compound) before being used to form the CBFV. However, this step assumes that the stoichiometric prevalence of the element is directly related to the contribution of the element to the compound's CBFV (and by extension, to the material property).

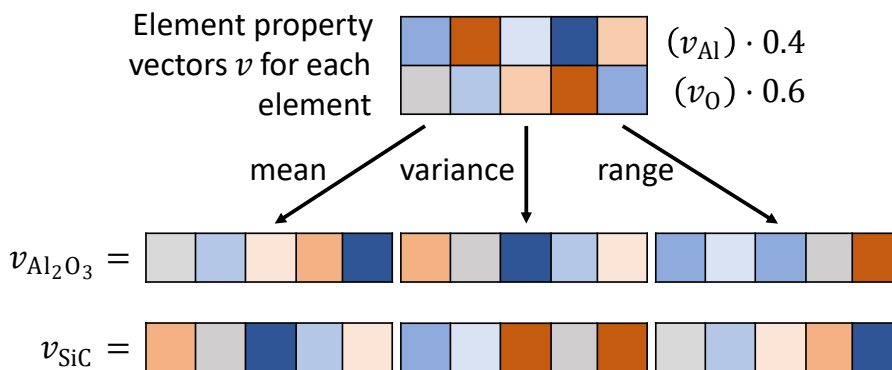


Figure 4.1: Example of the composition-based feature vector (CBFV) featurization of Al_2O_3 . The compound is featurized using the individual element vectors of aluminum and oxygen (v_{Al} and v_{O}), which are first weighted by the fractional prevalence of the elements. The featurized vector $v_{\text{Al}_2\text{O}_3}$ of Al_2O_3 is obtained by calculating the descriptive statistics (*e.g.*, mean, variance, and range) of the weighted element vectors. Another example featurized vector v_{SiC} of the compound SiC is also shown.

This is not true in all cases, an example being that of semiconductor doping. In this case, ML models using CBFV-featurized composition data cannot properly assign signif-

4.2 LACK OF STRUCTURE-AGNOSTIC DEEP LEARNING

icance to the dopant elements due to the insignificant contribution the dopant element vectors would make to the total compound vector. For example, in the CBFV featurization of the compound Al_2O_3 (Figure 4.1), the fractional prevalence of aluminum and oxygen are 0.4 and 0.6, respectively. To form the CBFV, the vectors representing aluminum and oxygen are scaled by a factor of 0.4 and 0.6, respectively. Descriptive statistics of these scaled vectors are then calculated and used to finally generate the full vector $v_{\text{Al}_2\text{O}_3}$ representing the compound Al_2O_3 .

In a typical doping scenario for semiconductors, however, the prevalence of dopant elements is much smaller compared to that of the host element. As an example, for the doping of silicon with phosphorus, changing the phosphorus concentration from 10^{12} cm^{-3} (around $2 \times 10^{-9} \text{ at\%}$) to 10^{21} cm^{-3} (about 2 at%) increases the electrical conductivity of the doped material by approximately eight orders of magnitude [108]! At such small dopant concentrations, the phosphorus vector will not make a significant contribution to the CBFV representing the doped compound—to the ML model, the CBFV of the doped silicon looks almost identical to the vector of pure silicon.

Moreover, the combination of all element vectors into one vector representing the compound decreases the expressiveness of the element vectors. For example, if the vector v_{Al} contains a positive value for a particular feature and the vector v_{O} contains a negative value for the same feature, then the corresponding sum of that particular feature in $v_{\text{Al}_2\text{O}_3}$ may be close to zero. While the choice of different descriptive statistics as mentioned above partially remediates this issue, it is still possible to encounter cases where the collapsing of multiple element vectors into one compound vector removes valuable information.

4.2 Lack of structure-agnostic deep learning

In addition to representing chemical compounds using the chemical formulae, it is also possible to include additional input data in the form of crystal structure information. For example, SchNet, CGCNN, MEGNet, DimeNet⁺⁺, and ALIGNN provide highly accurate material property predictions using graph representations of the chemical compound [109–113]. These graph-based modeling techniques incorporating structural information drastically outperform both traditional ML models utilizing CBFV inputs

as well as simpler DL architectures such as deep fully-connected neural networks [107, 114]. While these graph-based models are well-suited for studying and screening known materials with accompanying structural information such as those recorded in the Inorganic Crystal Structure Database (ICSD), Pearson’s Crystal Data (PCD), or other crystallographic or DFT databases, the reliance on structural information as input data severely limits the applicability of these models for novel materials discovery.

The number of crystal structures recorded in the ICSD and PCD are on the order of 10^6 and cover predominantly binary, ternary, and quaternary compositions with many common and duplicated structures [50, 102, 115, 116]. This number is exceedingly small when compared to the vast amount of possible chemical combinations in chemical whitespace (Section 2.1). When screening a large number of possible candidate compositions for a certain material property, it may be difficult to find a structural analogue for that composition in the available data.

Furthermore and perhaps more importantly, it is not possible to know the crystal structure of a novel material *a priori* when searching for novel materials, since these materials do not exist yet and cannot be characterized. Therefore, for the purposes of materials discovery, a “structure-agnostic” featurization approach which does not require crystal structure information is needed.

4.3 Transformers and the self-attention mechanism

To address the aforementioned issues, an improved featurization method for chemical compounds is required, which should (1) featurize the element properties and element prevalence of a compound separately, (2) preserve the identities of the elements throughout modeling, and (3) is structure-agnostic. Additionally, a new modeling technique is needed that can accept the input data from this improved featurization and use that data to make fast and accurate predictions for a wide range of materials properties.

These three requirements are achieved in this work by introducing the element-derived matrix (EDM) composition featurization method and by adapting the Transformer self-attention mechanism for the prediction of materials properties. The model implementing this mechanism is named the Compositionally Restricted Attention-Based network

4.3 TRANSFORMERS AND THE SELF-ATTENTION MECHANISM

(CrabNet). The Transformer self-attention mechanism as well as the novel EDM composition featurization method presented as part of CrabNet in this dissertation offer several benefits:

- General and robust learning framework for materials properties
- Explicit encoding for element identity and abundance
- Permutation-invariant composition representation
- Preservation of per-element property contributions
- Explicit element-element interactions
- Multi-head attention to observe different phenomena
- Increased interpretability of the model

CrabNet was written in a way such that it can be readily used to predict different types of materials properties for different applications. The model architecture and hyperparameters are chosen to enable rapid training and inference even in the absence of a graphics processing unit (GPU). Furthermore, the built-in uncertainty estimation, robust loss function, and flexible architecture allow CrabNet to accurately learn and predict material properties using datasets ranging from 312 to over 340 000 data records, without the need to specifically tailor the model. Further, the model performance of CrabNet matches or exceeds that of current state-of-the-art models.

In the novel EDM composition featurization scheme implemented by CrabNet, the encoding of an element's identity and its fractional abundance are separately performed. This leads to several advantages over the classical CBFV featurization:

- 1) Elements which exist in small amounts in chemical compositions are preserved in the EDM featurization using the fractional embedding and are not ignored in favour of the more abundant elements. Thus, pertinent information about trace elements (such as dopants) in the compounds are not discarded during featurization.
- 2) The element embedding can be taken from existing feature sets or learned as part of the model training, and is independent of the fractional embedding of the elements. This allows for more flexibility in tailoring the EDM for more specific material prediction tasks.

- 3) The constituent elements of a compound are encoded as separate embedding vectors in the EDM and are not collapsed into one vector as is in the case of the CBFV. This enables further modeling benefits as will be discussed in Chapter 5.
- 4) The use of multiple attention heads in the CrabNet architecture allows for the capturing of multiple types of element-element interactions for a given material property. This enables CrabNet to simultaneously capture multiple complex chemical relationships using the EDM to model materials phenomena that would otherwise be difficult to be learned using simpler architectures.

The second publication resulting from this thesis work discusses the aforementioned points in more detail and introduces the Compositionally Restricted Attention-Based network (CrabNet). The publication and its accompanying supplementary information (SI) are inserted in the following pages.

4.4 Publication 2: Compositionally restricted attention-based network for materials property predictions

Title	Compositionally restricted attention-based network for materials property predictions
Authors	A. Y.-T. Wang, S. K. Kauwe, R. J. Murdock, and T. D. Sparks
Journal	npj Computational Materials
Publisher	Nature Publishing Group
Publication date	May 28, 2021
Reference	<i>npj Computational Materials</i> , 2021 , 7: 77
DOI	10.1038/s41524-021-00545-1
Supporting information	The SI can be downloaded from: https://doi.org/10.1038/s41524-021-00545-1
Contributions	A.Y.-T.W. and S.K.K. jointly and in equal amounts conceived, developed the concept, and implemented the algorithms, code and visualizations described in this work. A.Y.-T.W. and S.K.K. jointly analyzed the results. R.J.M. assisted with developing the architecture and provided insight and guidance during model optimization and training. All authors discussed the results and contributed to the writing of the manuscript.

The article is inserted in the following pages. The article is reprinted under the open access CC BY license (<https://creativecommons.org/licenses/by/4.0/>).

ARTICLE OPEN



Compositionally restricted attention-based network for materials property predictions

Anthony Yu-Tung Wang^{1,3}, Steven K. Kauwe^{2,3}, Ryan J. Murdock² and Taylor D. Sparks²✉

In this paper, we demonstrate an application of the Transformer self-attention mechanism in the context of materials science. Our network, the Compositionally Restricted Attention-Based network (CrabNet), explores the area of structure-agnostic materials property predictions when only a chemical formula is provided. Our results show that CrabNet's performance matches or exceeds current best-practice methods on nearly all of 28 total benchmark datasets. We also demonstrate how CrabNet's architecture lends itself towards model interpretability by showing different visualization approaches that are made possible by its design. We feel confident that CrabNet and its attention-based framework will be of keen interest to future materials informatics researchers.

npj Computational Materials (2021)7:77; <https://doi.org/10.1038/s41524-021-00545-1>

INTRODUCTION

Materials scientists constantly strive to achieve better understanding, and therefore better predictions, of materials properties. This began with the collection of empirical evidence through repeated experimentation, resulting in mathematical generalizations, theories, and laws. More recently, computational methods have arisen to solve a large variety of problems that were intractable to analytical approaches alone^{1,2}.

As experimental and computational methods have become more efficient, high-quality data has opened up a new avenue to materials understanding. Materials informatics (MI) is the resulting field of research that utilizes statistical and machine learning (ML) approaches in combination with high-throughput computation to analyze the wealth of existing materials information and gain unique insights^{2–4}. As this wealth has increased, practitioners of MI have increasingly turned to deep learning techniques to model and represent inorganic chemistry, resulting in approaches such as ElemNet, IRNet, CGCNN, SchNet, and Roost^{5–9}. In specific cases including CGCNN and SchNet, the compounds are represented using their chemical and structural information^{7,8,10–15}.

Modeling approaches based on crystal structure are an excellent tool for ML. Unfortunately, there are many material property datasets that lack suitable structural information. An example of this is the experimental band gap data gathered by Zhou et al.¹⁶. Conversely, many databases such as the Inorganic Crystal Structure Database (ICSD) and Pearson's Crystal Data (PCD) contain an abundance of structural information, but lack the associated material properties of the recorded structures. In both cases, the applicability of structure-based learning approaches are limited. This limitation is particularly evident in the discovery of novel materials, since it is not possible to know the structural information of (currently undiscovered) chemical compounds a priori. Therefore, the development of structure-agnostic techniques is well-suited to the discovery of novel materials.

A typical approach to structure-agnostic learning is done by representing chemistry as a composition-based feature vector (CBFV)¹⁷. This allows for data-driven learning in the absence of structural information. The CBFV is a common way to transform chemical compositions into usable features for ML and is

generated from the descriptive statistics of a compound's constituent element properties. Researchers have effectively used CBFV-based ML techniques to generate material property predictions^{17–25}.

One potential issue with the CBFV approach lies in the way the element vectors are combined to form the vector describing the chemical compound. Typically, the individual element vectors of the compound are scaled by the element's prevalence (fractional abundance) in the composition, before being used to form the CBFV. This step assumes that the stoichiometric prevalence of constituent elements in a compound dictates their chemical signal, or contribution, to the material's property. However, this is not true in all cases; an extreme example of this is element doping. Dopants can be present in very small amounts in a compound, but can have a significant impact on its electrical^{23,26,27}, mechanical^{20,28–30}, and thermal properties^{31–34}. In the case of a typical CBFV approach that uses the weighted average of element properties as a feature, the signal from dopant elements would not significantly change the vector representation of a compound. As a result, the trained ML model would fail to capture a portion of the relevant chemical information available in the full composition.

It is apparent that there is no generally accepted best way to model materials property behaviors. Different ML approaches lend themselves towards different modeling tasks. CGCNN requires access to structural information, ElemNet operates within the realm of large data, and classical models excel when domain knowledge can be exploited to overcome data scarcity³⁵. To address the diversity of learning challenges, in Dunn et al., the Automatminer framework uses computationally expensive searches to optimize classical modeling techniques. They demonstrate effective learning on some data, but show shortcomings when deep learning is appropriate³⁶.

In a similar spirit, we seek to overcome general challenges in the area of structure-agnostic learning using an approach we refer to as the Compositionally Restricted Attention-Based network (CrabNet). CrabNet introduces the self-attention mechanism to the task of materials property predictions, and dynamically learns and updates individual element representations based on their chemical environment. To enable this, we introduce a

¹Fachgebiet Keramische Werkstoffe/Chair of Advanced Ceramic Materials, Technische Universität Berlin, Berlin, Germany. ²Department of Materials Science & Engineering, University of Utah, Salt Lake City, UT, USA. ³These authors contributed equally: Anthony Yu-Tung Wang, Steven K. Kauwe. ✉email: sparks@eng.utah.edu

featurization scheme that represents and preserves individual element identities while sharing information between elements. Self-attention is a procedure by which a neural network learns representations for each item in a system based on the other items that are present. In this context, we treat the chemical composition as the system and the elements as the items within that system. This representation enables *CrabNet* to learn inter-element interactions within a compound and use these interactions to generate property predictions.

To perform self-attention, we use the Transformer architecture, which emerged from natural language processing (NLP) and is based on stacked self-attention layers^{37–42}. A typical use of the Transformer architecture in NLP is to encode the meaning of a word given the surrounding words, sentences, and paragraphs. Beyond NLP, other example uses of the Transformer architecture are found in music generation⁴³, image generation⁴⁴, image and video restoration^{45–49}, game playing agents^{50,51}, and drug discovery^{52,53}. In this work, we explore how our attention-based architecture, *CrabNet*, performs in predicting materials properties relative to the common modeling techniques *Roost*, *ElemNet*, and random forest (RF) for regression-type problems.

RESULTS

The results of this study are described in three subsections. First, we describe the collection of materials property data used for benchmarking *CrabNet*. Second, we highlight the performance of *CrabNet* when compared to other current learning approaches which rely solely on composition. Third, we briefly outline how the self-attention mechanism in *CrabNet* enables visualizations and inspectability unique to attention-based modeling.

Data and materials properties procurement

For this work, we obtained both computational and experimental materials data for benchmarking. Our benchmark data includes materials properties from the Matbench dataset as provided by Dunn et al.³⁶. In addition, materials properties data from a number of works^{6,54–57} are collected, which are referred to as the Extended dataset. We included 28 benchmark datasets in total: 10 from the Matbench and 18 from the Extended datasets ranging from 312 to 341,788 instances of data.

The Matbench datasets were split using fivefold cross-validation following instructions provided in the original publication³⁶. Materials properties in the Extended dataset were split into train, validation, and test datasets using a fixed random seed. For both datasets, several steps were taken to process the original datasets to be compatible with structure-agnostic learning using *CrabNet*. Care was taken to ensure that (1) no duplicate compositions were present in each of the train, validation, and test datasets, and that (2) if a composition exists in the train or validation dataset, all compounds with the same composition are removed from the validation and test datasets. To remain comparable with the Automaterminer publication³⁶, we applied the data processing steps as mentioned above after splitting the data. Please note that since some datasets have more duplicate compositions than others, these processing steps may affect the train/val/test ratios. For duplicate compositions in the OQMD and MP datasets, the target value associated with the lowest formation enthalpy was selected. For other datasets, the mean of the target values was used. Please see the Supplementary Methods for more details.

The full processed benchmark dataset, comprising the Matbench and Extended datasets, were then used with *Roost*, *CrabNet*, *ElemNet*, and RF models. The training and validation data were used for training and hyperparameter tuning. The test data were held-out to provide a fair evaluation of performance metrics across all models. Model performance was only evaluated

after all training and hyperparameter tuning was completed. A summary of the datasets is shown in Table 1. All datasets are provided as pre-split csv files to facilitate future comparisons to the metrics reported in this paper. Additional data processing and cleaning details can also be seen in the code on the dataset repository *mse_datasets*⁵⁸. To maintain consistent and simple benchmark comparisons, we selected data suitable for regression tasks and ignored structural information when present.

Benchmark comparisons

With the benchmark data described above, we generated material predictions using the publicly available code repositories for *Roost*⁹, *CrabNet*⁵⁹, and *ElemNet*⁵. The performance of these benchmarked models is compared using the mean absolute error (MAE) between n true values (y) and predicted values (\hat{y}) as defined by Eq. (1):

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|. \quad (1)$$

This allows for consistent comparison to past works^{5–7,9}.

Figure 1 shows the performance metrics from training and testing the models on all the benchmark materials properties outlined above. Here we note that the models for *Roost*, *CrabNet*, and *ElemNet* were all trained using the default model parameters provided with their respective repositories. In contrast to *Roost* and *ElemNet*, the default parameters for *CrabNet* were optimized using validation data from some of the same datasets on which we benchmarked. Although it is possible this offers a small advantage to *CrabNet*'s performance, we do not expect this to be significant due to *CrabNet*'s consistently strong performance on all benchmark tasks.

We tested two versions of *CrabNet*. The default *CrabNet* uses a *mat2vec* embedding when representing elements, similar to *Roost*. The second version of *CrabNet* (*HotCrab*) uses one-hot encodings (in the form of atomic numbers) and fractional amounts to represent each element in a composition. This is similar to *ElemNet*, as both models start without any chemical information. The random forest (RF) model utilizes a Magpie-featurized CBFV to represent chemistry. This is included as a performance baseline to match similar works^{5,9,36}.

Overall, we see similar performance between *Roost* and the two versions of *CrabNet* tested. Given the different architectures and modeling philosophies of *Roost* and *CrabNet*, it is promising that both approaches converge towards the same performance metrics. We also see that *Roost* and both *CrabNet* versions achieve consistent and significant improvements to MAE compared to *ElemNet* and RF approaches. Interestingly, Fig. 1 shows that the use of *mat2vec* instead of one-hot with *CrabNet* improves prediction performance on all materials properties except for AFLOW thermal conductivity, MP elastic anisotropy, and those present in the largest datasets (OQMD).

The Matbench data provided by Dunn et al.³⁶ was benchmarked using the Automaterminer tool. These metrics are not included in Fig. 1, since all but two (*expt_gap*, and *steels_yield*) of Automaterminer's models use structural information. Consequently, we focus on these two materials properties when comparing *CrabNet*'s results to those from Automaterminer. For these two metrics, *CrabNet*'s structure-agnostic approach outperforms the reported MAE values from Automaterminer on the same tasks (*expt_gap*: 0.416 eV vs. 0.338 eV for *CrabNet*; *steels_yield*: 95.2 GPa vs. 91.7 GPa for *CrabNet*).

The performance of *CrabNet* on the *steels_yield* task is particularly interesting. The *steels_yield* dataset contains compounds with small dopant amounts in large chemical systems (up to 13 elements per composition) and only 312 total data. *CrabNet*'s ability to learn on this data-poor property and outperform all other tested models including the baseline RF

Table 1. Benchmark datasets. List of all 28 material properties used to benchmark the ML models in this work, together with the dataset size and the original training, validation, and test set proportions.

Dataset name	Source	Material property	# Samples	(Train/val/test) %
castelli	Castelli et al. ^{36,65}	Formation enthalpy (perovskites)	18928	5-fold (72/8/20)
dielectric	MP ^{36,66–68}	Refractive index	4764	5-fold (72/8/20)
elasticity_log10 (G_VRH)	MP ^{36,66,67,69}	log ₁₀ (shear modulus (VRH))	10987	5-fold (72/8/20)
elasticity_log10 (K_VRH)	MP ^{36,66,67,69}	log ₁₀ (bulk modulus (VRH))	10987	5-fold (72/8/20)
expt_gap	Experiment ^{16,36}	Experimental band gap	4764	5-fold (72/8/20)
jdft2d	Experiment ^{36,70}	Exfoliation energy	636	5-fold (72/8/20)
mp_e_form	MP ^{36,66,67}	Formation energy per atom	132752	5-fold (72/8/20)
mp_gap	MP ^{36,66,67}	Band gap	106113	5-fold (72/8/20)
phonons	MP ^{36,66,67,71}	Phonon frequency	1265	5-fold (72/8/20)
steels_yield	MP ^{36,72}	Steels yield strength	312	5-fold (72/8/20)
aflow_ael_bulk_modulus_vrh	AFLOW ⁵⁴	Bulk modulus (VRH)	4905	(70/15/15)
aflow_ael_debye_temperature	AFLOW ⁵⁴	Debye temperature	4905	(70/15/15)
aflow_ael_shear_modulus_vrh	AFLOW ⁵⁴	Shear modulus (VRH)	4905	(70/15/15)
aflow_agl_thermal_conductivity_300K	AFLOW ⁵⁴	Thermal conductivity	4896	(70/15/15)
aflow_agl_thermal_expansion_300K	AFLOW ⁵⁴	Thermal expansion	4895	(70/15/15)
aflow_Egap	AFLOW ⁵⁴	Band gap	27841	(70/15/15)
aflow_energy_atom	AFLOW ⁵⁴	Energy per atom	27844	(70/15/15)
CritExam_Ed	Bartel et al. ⁵⁵	Decomposition enthalpy	85014	(70/15/15)
CritExam_Ef	Bartel et al. ⁵⁵	Formation enthalpy	85014	(70/15/15)
mp_bulk_modulus	MP (Oct. 2018) ^{57,66,67,69}	Bulk modulus	7632	(70/15/15)
mp_elastic_anisotropy	MP (Oct. 2018) ⁵⁷	Ratio of elastic anisotropy	7659	(70/15/15)
mp_e_hull	MP (Oct. 2018) ⁵⁷	Energy above the convex hull	83983	(70/15/15)
mp_mu_b	MP (Oct. 2018) ⁵⁷	Magnetization of the unit cell	83973	(70/15/15)
mp_shear_modulus	MP (Oct. 2018) ^{57,66,67,69}	Shear modulus	7437	(70/15/15)
OQMD_Bandgap	OQMD ⁶	Band gap	341696	(70/15/15)
OQMD_Energy_per_atom	OQMD ⁶	Energy per atom	341788	(70/15/15)
OQMD_Formation_Enthalpy	OQMD ⁶	Formation enthalpy	341788	(70/15/15)
OQMD_Volume_per_atom	OQMD ⁶	Volume per atom	341788	(70/15/15)

The materials properties listed in the top and bottom halves are Matbench and Extended datasets, respectively.

model (which is traditionally better in the data-poor regime) is encouraging. We expected the steels_yield task to be difficult for all deep learning approaches. Nevertheless, repeated training and validation of CrabNet consistently produced error metrics better than the best result obtained by Automatminer (95.2 GPa).

Visualizing self-attention

CrabNet's modeling and visualization capabilities are enabled by its attention-based learning framework. In statistical ML and many deep learning approaches akin to ElemNet, the chemical composition of a compound is represented as a single CBEV. In contrast, Roost and CrabNet represent a composition as a set of element vectors. Distinct to CrabNet, however, is the Transformer-based self-attention mechanism that learns to update these element vectors using learned attention matrices. In Fig. 2, we show example attention matrices for each attention head of a CrabNet model trained on the property mp_bulk_modulus, using Al₂O₃ as the example composition. These matrices contain information regarding how each element (rows) is influenced by all other elements in the system as well as itself (columns). The values in these attention matrices are used in the Transformer encoder to update the element vectors. A value of zero means that the element in the column is completely ignored when updating the element in that row. A value of one means that the entire vector update is based solely on that column's element.

Our implementation of CrabNet has three layers, each with four attention heads, with each head using the same data to generate its own independent attention matrix (see "Methods" for more details).

Shifting our focus to another CrabNet model trained on aflow_Egap data, we show that in addition to visualization of the individual attention heads, we can also generate a global view of attention from the perspective of individual elements. In Fig. 3, we use four periodic tables to visualize, for each attention head, the average attention that silicon dedicates to other elements when they are in the same composition. The darker colored elements can be understood as highly influential when updating silicon's vector representation.

Interestingly, each attention head exhibits its own behavior, with some focusing on familiar groups and columns in the periodic table. This behavior lends credibility to CrabNet since there is no inherent reason that data-driven learning should converge to chemical rules that are familiar to materials scientists. Furthermore, the identification of unfamiliar element groupings enabled by the attention-based visualizations may allow us to formulate further research questions to study these inter-elemental interactions.

The preservation of elemental identity within a compound—as a result of the self-attention mechanism—also enables CrabNet to generate property predictions in a way that is different to other approaches shown in the literature. Typically, element information

a) Matbench Properties					
	Roost	CrabNet	HotCrab	ElemNet	RF
Castelli perovskites	0.148	0.127	0.135	0.194	0.152
Refractive index	0.370	0.348	0.366	0.442	0.476
Shear modulus (log10)	0.100	0.092	0.097	0.125	0.100
Bulk modulus (log10)	0.073	0.068	0.072	0.090	0.078
Experimental band gap	0.373	0.338	0.352	0.439	0.447
DFT Exfoliation energy	52.879	50.512	54.811	61.714	55.105
MP Formation energy	0.078	0.077	0.080	0.746	0.131
MP Band gap	0.260	0.263	0.273	0.313	0.363
Phonon peak	49.767	53.341	60.253	nan	68.687
Steels yield	155.190	91.748	92.479	nan	103.898
b) Extended Properties					
	Roost	CrabNet	HotCrab	ElemNet	RF
AFLOW Bulk modulus	8.820	8.692	9.103	12.119	11.907
AFLOW Debye temperature	37.167	33.464	35.755	45.723	36.484
AFLOW Shear modulus	9.983	9.082	9.430	13.319	10.094
AFLOW Thermal conductivity	2.703	2.318	2.254	3.322	2.658
AFLOW Thermal expansion	3.69e-06	3.85e-06	3.88e-06	5.42e-06	5.44e-06
AFLOW Band gap	0.337	0.301	0.316	0.372	0.384
AFLOW Energy per atom	0.086	0.093	0.094	0.122	0.224
Bartel Decomposition (Ed)	0.067	0.063	0.066	0.079	0.076
Bartel Formation (Ef)	0.055	0.059	0.059	0.071	0.100
MP Bulk modulus	11.395	11.209	11.918	15.136	14.358
MP Elastic anisotropy	8.082	8.263	8.126	8.191	11.691
MP Energy above convex hull	0.094	0.089	0.092	0.110	0.126
MP Magnetic moment	2.507	2.105	2.180	2.694	2.732
MP Shear modulus	12.797	12.787	12.849	15.097	12.777
OQMD Band gap	0.088	0.049	0.048	0.148	0.060
OQMD Energy per atom	0.032	0.033	0.033	0.062	0.141
OQMD Formation enthalpy	0.032	0.031	0.031	0.049	0.083
OQMD Volume per atom	0.296	0.277	0.278	0.442	0.544

Fig. 1 Benchmark results. MAE scores of Roost, CrabNet, one-hot encoded CrabNet (HotCrab), and ElemNet on the held-out test datasets, compared with the random forest (RF) baseline for (a) the Matbench dataset and (b) the Extended dataset. Cells are colored according to relative MAE performance within each row (blue is better, and red is worse). A NaN (not a number) value is reported for instances where the models failed to converge on a given material property. Here we present model results trained using chemical information (Roost, CrabNet), no chemical information (HotCrab, ElemNet), and a standard CBFV (RF).

of a given compound is collapsed into a single vector first and then used to generate the property prediction. In contrast, CrabNet uses each element's vector representation to directly predict the element's contribution to the property prediction. Figure 4a shows the average contributions from each element for a CrabNet model trained on AFLOW bulk modulus data. The darker colored elements contribute more towards a compound's bulk modulus value. Alternatively, elements can be visualized individually using their specific per-element contributions. In Fig. 4b we show distribution plots for lithium and tungsten's contributions to bulk modulus. From these plots, we can see that CrabNet expects lithium to contribute little to the overall bulk modulus, whereas it expects tungsten to contribute largely. See Supplementary Fig. 3 for additional examples of these element contribution plots. The visualizations from Fig. 4 match closely—and reinforce—expectations regarding which elements most influence bulk modulus behavior in a compound. Exploration of data in this manner hints at the first steps towards model interpretability of CrabNet. We expect these types of property visualizations to be useful for exploring and verifying model and element behavior in detail.

Finally, with per-element contributions in mind, we can demonstrate changes to CrabNet's expected material property behavior as a function of chemical composition. To do this, we consider a normalized chemical system consisting of atoms A and B, in the form of A_xB_{1-x} . We then generate property predictions for all $x \in \{0.0, 0.02, \dots, 1.0\}$. In Fig. 5, we visualize CrabNet's behavior when predicting band gap of the Si_xO_{1-x} system using a model trained on the `aflow_Egap` data.

We first observe that the expected elemental contributions for both oxygen and silicon to band gap are similar throughout the varied stoichiometry range, with the exception of the peak in oxygen contribution at around $x = 0.7$. We also observe that the

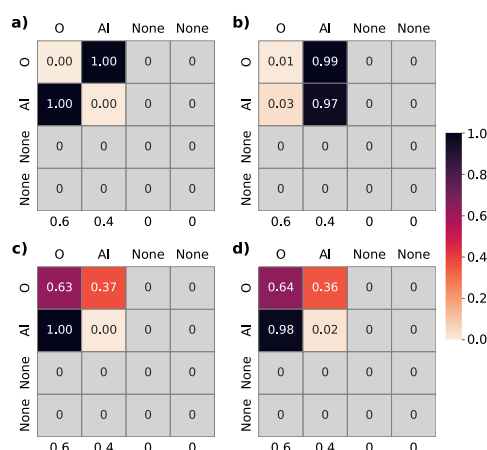


Fig. 2 Visualization of self-attention in one compound. Displayed are the four attention heads (a–d) from the first layer of a CrabNet model trained on `mp_bulk_modulus` and evaluated on the composition Al_2O_3 . Each row represents an element in the system. Each column represents an element being attended to. Each element's fractional amount is shown on the x-axis. The values in the attention matrix are scores representing element–element interactions for the compound. As an example, in head a, $Al_{0.4}$ and $O_{0.6}$ are attending strongly to each other, with attention scores of 1.00 between these two elements.

model indicates a transition of the Si_xO_{1-x} system between conducting and semi-conducting between $x = 0.5$ and $x = 0.7$. We note that the only available training data sample from the Si_xO_{1-x} system in the dataset was from the composition SiO_2 . Therefore, we can see that the band gap trend predicted here by CrabNet is based on the learned chemical representations and inter-elemental interactions from other elements and systems. The visualization of CrabNet model predictions within a given chemical space is an alternative way to explore model learning and prediction behavior, and may lead to an improved understanding of inter-elemental interactions within a chemical system.

Furthermore, we note that the ability of CrabNet in predicting material property trends for specific chemical systems without requiring a large amount of training data for that system is of great benefit. For future studies, this ability may be investigated for its application in predicting the behavior of new chemical systems while only requiring a sparse sampling or learning of their chemical information. Furthermore, we believe that transfer learning of trained CrabNet models to other material properties is possible, due to the ability of the self-attention mechanism to accurately capture inter-elemental interactions. We are confident that these ideas of probing and visualizing of CrabNet's modeling process and model predictions will open up further interesting research directions and ultimately lead to more insights in the pursuit of inspectable models.

DISCUSSION

Unique challenges exist when applying ML to materials science. In this paper, we address the limitations of ML on chemical composition by introducing CrabNet. The CrabNet architecture uses the self-attention mechanism and the EDM representation scheme to perform context-aware learning on materials properties. Using 28 benchmark datasets, we demonstrate CrabNet's performance compared to Roost, ElemNet, and RF baselines.

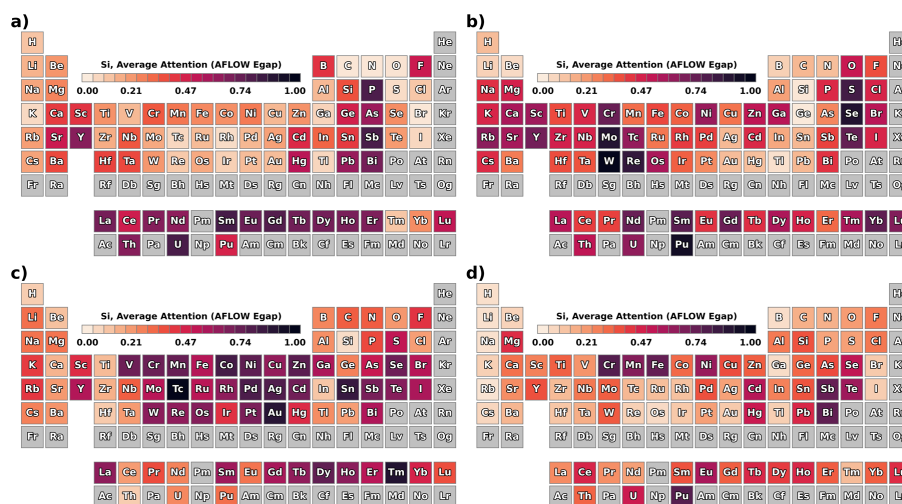


Fig. 3 Visualization of average attention for one dataset. The average attention from each of the four attention heads (a–d) from the first layer of a CrabNet model trained on the aflow_Egap data is shown for systems containing Si. The heatmap shows the average amount of attention that Si dedicates to the other elements in Si-containing compounds. The darker the coloring, the more strongly Si attends to that element. We can see that each attention head exhibits its own behavior, and attends to different groups of elements. Interestingly, head **a** attends to common *n*-type dopants and head **c** attends to many transition metals, whereas heads **b** and **d** have unfamiliar element groupings.

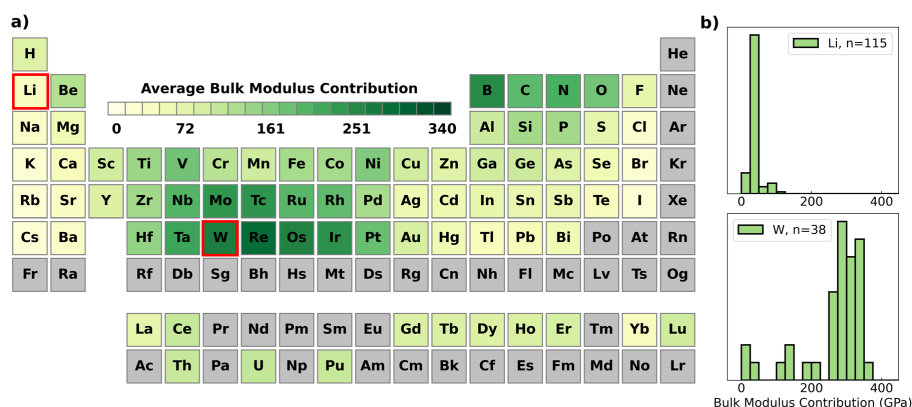


Fig. 4 Overall element contribution to property predictions. Average contribution of all elements to bulk modulus predictions, computed from the AFLOW bulk modulus dataset, (a) plotted on a periodic table and (b) plotted as a distribution showing the per-element contribution amounts of Li and W, respectively, in all the compounds. The darker colored elements in the periodic table contribute more towards a compound's bulk modulus value.

CrabNet exhibits consistent predictive accuracy across the full range of materials properties tested. Furthermore, we show that the self-attention-based learning technique also provides alternative methods for visualizing model behavior. We demonstrate the use of attention and per-element contribution prediction capabilities for visualizing common trends in our trained models that match chemical expectations.

Given this application of self-attention in the context of materials science, we expect that there can be many informative and impactful follow-up works. Specifically, we believe these will largely fall into three thematic categories:

1. CrabNet directly contributing to the community's focus towards improved property predictions.

CrabNet consistently generates good MAE scores. The performance achieved with the use of self-attention, combined with the innovative use of element and composition featurization techniques, will allow researchers to delve deeper into analyzing and predicting materials properties. As a result, we believe that CrabNet will be relevant in areas where other ML methods fall short (e.g., dopants, small data, and materials extrapolation tasks). We also note that with minimal changes to CrabNet, it can also perform classification tasks; we expect CrabNet to

similarly excel at this.

2. Attention-based models allow for new ways of thinking about materials-specific problems.

In this work, we briefly examined the attention mechanism. Attention highlights important interactions and may be used to understand which element interactions mediate materials properties. Model explainability has thus far been elusive to the traditional ML paradigms; the inclusion of self-attention in this work has introduced additional methods of model inspectability that may be a step towards this goal.

3. Augmentation of *CrabNet* using structural and domain-specific knowledge.

This work intentionally used a compositionally restricted EDM representation with no structural information. Structure-agnostic learning is an important task in ML and *CrabNet* demonstrates that accurate learning is achievable using the self-attention mechanism. However, the prediction of materials properties using structural information is also an important task. Integration of structural information could be achieved by describing elements in their structural and chemical environments. We expect that the self-attention mechanism of *CrabNet* will be able to utilize this additional information to

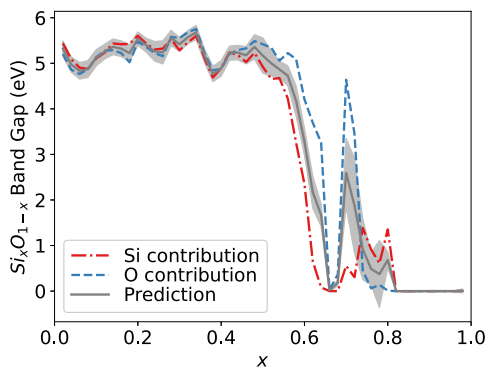


Fig. 5 Element contribution to property prediction as a function of composition. Model predictions over the $\text{Si}_x\text{O}_{1-x}$ system using a model trained on the aflow_Egap data. The x -axis is the fractional amount of Si. The y -axis shows the predicted band gap value at a given composition. The blue and red lines are the individual element contributions to the prediction, as predicted by *CrabNet*. The gray shading represents the aleatoric uncertainty for each prediction.

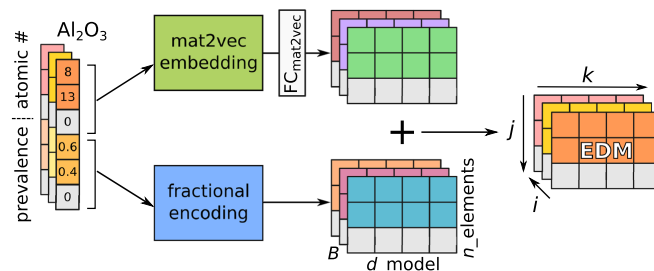


Fig. 6 EDM featurization scheme. Schematic illustration of the element-derived matrix (EDM) representation for Al_2O_3 , where B represents the batch, d_{model} is the element features, and n_{elements} represents the number of elements. Composition slices, when concatenated across batch dimension i , form an EDM tensor which is then used as the model input to *CrabNet*. When a chemical formula has fewer elements than rows in the EDM, the extra data rows are filled with zeros.

4.4 PUBLICATION 2: CRABNET

make more accurate predictions. This application of attention-based learning to crystal systems is an exciting and promising direction. We also expect that materials prediction tasks involving processing steps or other non-compositional features could be used in this approach. Both of these changes could easily be implemented as extensions to the EDM.

While further research is necessary to fully discern the utility of self-attention in materials problems, we believe that this paper highlights a major new direction in its application in ML and suggests exciting directions for future research.

METHODS

Self-attention and the *CrabNet* architecture

Chemical compositions are input using the atomic numbers and fractional amounts of their constituent elements. The atomic numbers are used to retrieve element representations (either *mat2vec* or *one-hot*). The fractional amounts are used to obtain fractional embeddings (described below). An element embedding matrix is generated by applying a fully connected network to the element representations. A fractional embedding matrix is created from the fractional embeddings. These matrices are then added together (element-wise) to generate the element-derived matrix (EDM, see Fig. 6). Each row of the EDM (j -index) represents an element and the columns (k -index) contain the element embeddings. We batch each unique chemical composition onto a third dimension (the i -index). The resulting three-dimensional tensor contains the input data for the *CrabNet* architecture.

We use the *mat2vec* element embeddings⁵⁰ as the default source of chemical information for each element, even though there are other choices for element properties available, such as *Jarvis*²², *Maggie*⁶¹, *Oliynyk*¹⁸ or a simple *one-hot* encoding. The *mat2vec* embedding has the advantage of being pre-scaled and normalized, and having no missing elements nor element features. Regardless of the choice of element representation, the representation must be reshaped to fit the attention input dimensions of (d_{model}). This is done using a learned embedding network; the result is a matrix of size ($n_{\text{elements}}, d_{\text{model}}$). In addition to the default training of *CrabNet* using the *mat2vec* embedding, a *one-hot* embedding of the elements was used to train an additional *CrabNet* model (*HotCrab*) to better facilitate comparison with *ElemNet*.

The stoichiometric information for each element in the EDM is represented by two fractional embeddings. The fractional embeddings are inspired by the positional encoder as described in the seminal work by Vaswani et al.³⁷. We use sine and cosine functions of various periods to project the fractional amounts into a high-dimensional space (dimension $d = d_{\text{model}}/2$) where smooth interpolation between fractional values is preserved. The first part of the fractional embedding represents the stoichiometry, using the normalized fractional amounts, on a linear scale with a fractional resolution of 0.01. The second part of the embedding maps stoichiometry using a log scale and spans from 1×10^{-6} to 1×10^{-1} . This logarithmic transformation of the fractional embedding preserves small fractional amounts such as those present in doping. The two parts of the fractional embedding for all elements are concatenated across the

Parameter	Description	Default value
d_{in}	(Input) dimension of element embedding	200 (mat2vec); 118 (one-hot)
d_{model}	Dimension for EDM and positional encoder	512
d_{ff}	Feedforward dimension for self-attention mechanism	2048
d_k	Key dimension (equal to d_q in this work)	$d_{model}/H = 128$
H	Number of attention heads per attention block	4
N	Number of stacked self-attention layers	3
$node_{res}$	Number of nodes at each layer for residual network	[1024, 512, 256, 128]
d_{out}	(Output) dimensions of residual network	3

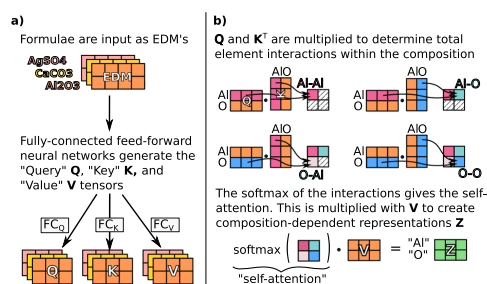


Fig. 7 Schematic of an attention block in the *CrabNet* architecture. **a** The initial projection of the input EDM into the Q , K and V tensors. **b** The scaled dot-product attention operation obtaining the self-attention matrix and the updated Z element representation. The batch dimension is not shown in **b** to improve legibility.

embedding dimension to obtain a matrix of size $(n_{elements}, d_{model})$. See Supplementary Figs. 1 and 2 for example visualizations of the EDM embedding.

Once the element and fractional embeddings are calculated and added together, we then batch the finished EDMs across the first dimension. This gives the final input data of shape $(n_{compounds}, n_{elements}, d_{model})$, where $n_{compounds}$ is the total number of compounds in a given batch, $n_{elements}$ is the number of rows in the EDM (inferred from the number of elements in the largest composition in a given dataset), and d_{model} is the size of the embeddings. Here, we also note that the exact ordering of the element rows (j) in a compound in the EDM does not influence *CrabNet* due to the permutation-invariant nature of the self-attention mechanism.

CrabNet contains two primary modules with the default hyperparameters as shown in Table 2. The first module is a Transformer encoder with 3 layers and 4 attention heads in each layer. The second module is a residual network that converts element vectors into element contributions.

To understand the Transformer encoder, we first describe the self-attention mechanism. During self-attention (Fig. 7a), the EDM is operated on by three fully connected linear networks (FC_Q , FC_K , and FC_V). These networks generate the query Q , key K , and value V tensors. These tensors can be conceptualized as a learned high-dimensional space where the model stores chemical behavior from the training data.

The K and Q tensors contain information regarding the magnitude to which elements interact. The V tensor stores the information that is used to map from element to property contribution. The dot product of each Q and K^T tensor pair (where K^T denotes the transpose of K) generates the relative element importances in the system (Fig. 7b). The importances are scaled using a constant $\sqrt{d_k}$ and then normalized using a softmax function. This results in the self-attention tensor, commonly referred to as the attention map. We denote this tensor as A . The matrix multiplication of A with V updates the element-representations in the compound based on the importance of each element.

Each of the four attention heads independently performs self-attention with their own Q_h , K_h , V_h , and Z_h tensors, where h denotes the head index for $h = 1, \dots, H$. As a result, the network generates four different element representations at each layer. The individual Z_h tensors are concatenated across the last dimension to make the Z tensor (as seen in Fig. 8a). The Z tensor is then passed into a linear FC network which combines the element representations from each head. The output of this FC network is an updated EDM' (for each composition in the batch). This process of converting an EDM into an updated EDM' is referred to as a self-attention block. *CrabNet* repeats the process of updating the EDM via the self-attention block three times (hence, three layers) resulting in the final updated representations, denoted EDM''. This concludes the Transformer encoder module.

Once the Transformer encoder has updated the element representations, each EDM'' passes through a fully connected residual network with hidden layer dimensions of $node_{res}$. The residual network then transforms the EDMs into the shape $(n_{elements}, n_{elements}, 3)$. We define these final three vectors as the element-proto-contributions p' , element-uncertainties u' , and element-logits (see Fig. 8a). The element scaling factor s is obtained by taking the sigmoid (σ) of the element-logits. The element-contributions are then obtained by multiplying the element-proto-contributions p' by their respective scaling factor s . This results in element contributions y' . Finally, the mean of the element contributions is taken and output as the predicted property value for each compound (see Fig. 8b). Similarly, the mean of the element-uncertainties is used in the aleatoric uncertainty prediction as described by Roost⁹.

Training *CrabNet*

After the featurization of compositions into EDMs, the dataset loading and batching is performed with the built-in *Datasets* and *DataLoaders* classes from *PyTorch*. All target values are scaled to zero-centered mean and unit variance for training and inference. The target scaling is then undone for performance evaluation. Batch size during training is dynamically calculated using the training set size for faster training, and limited to be within the range 2^7 – 2^{12} . For inference, the batch size was fixed at 2^7 .

Model weights are updated using the look-ahead⁶² and Lamb optimizer⁶³ with a learning rate that is cycled between 1×10^{-4} and 6×10^{-3} every 4 epochs to achieve consistent model convergence. A robust MAE is used as the loss criterion for model performance⁹. The default parameters generalize well when predicting most of the benchmark materials properties. Although we expect that optimization of hyperparameters may improve *CrabNet*'s results for individual materials properties, we believe it is more important that materials scientists be able to use *CrabNet* with little or no adjustments to the underlying code.

It is a known phenomenon that random weight initialization can impact the performance of the Transformer encoder architecture. Thus, to mitigate variance in the performance metrics between different model runs, we trained *CrabNet* using a fixed random seed of 42 for all training runs across all materials properties. We do note that in the case of random model initialization, the run-to-run variation between different trained models is a feature that could be taken advantage of for determining the epistemic uncertainty. Unfortunately, due to the sheer volume of materials properties investigated in this work and the limited compute resources available, we have not investigated this thus far.

Finally, we note that all model training, evaluation, and benchmarking (for *CrabNet*, Roost, ElemNet, and RF) was conducted on a single

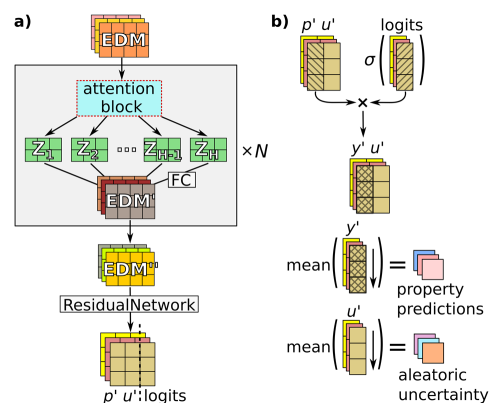


Fig. 8 Overall CrabNet architecture and prediction of material property and uncertainty. **a** Schematic of the CrabNet architecture including the input EDM, the self-attention layers (repeated N times), the updated and final element representations (EDM' and EDM''), the residual network, and the final model output. **b** The calculation steps for element contributions and prediction of the targets and uncertainties. The p' and u' vectors represent the element-proto-contributions and the element uncertainties, respectively. y' represent the element contributions. The material property is obtained by taking the mean of element-contributions (y') for each compound. Similarly, the mean of the element-uncertainties (u') gives us the estimated aleatoric uncertainty.

workstation PC equipped with an Intel i9-9900K CPU, 32 GB of DDR4 RAM, and two NVIDIA RTX 2080 Ti GPUs with 11 GB VRAM per GPU. The deep learning models were trained on the GPU, while the RF models were trained on the CPU.

Reference models

Predictions for all materials properties were generated using code from the Roost repository⁷. Minor adaptations were made to the code to allow for automated training and benchmarking. Overall, Roost generates consistently impressive results. Roost relies on a soft-attention mechanism used over a graph representation of the compound. This is in the same spirit of CrabNet, and both seek to generate vector representations for the elements in the system without using structure information. The residual network and robust loss function from Roost were helpfully adopted into our architecture⁹.

Predictions from ElemNet were generated using default parameters using code from the repository³. Custom scripts were written to train and evaluate ElemNet over all materials properties data. ElemNet consistently under-performed compared to Roost and CrabNet. ElemNet failed to converge for multiple properties resulting in NaN (not a number) values in the model outputs. Examples of this occurring can be seen in the phonons and steels yield datasets. Here, we would like to note that IRNet⁶ could also be benchmarked and compared in this study. However, due to the prohibitively large computational requirements, we chose not to train and evaluate IRNet. We do however note that the OQMD performance reported in the IRNet publication⁶ is consistently lower than both Roost and CrabNet for the same properties. These following values show the reported performance of IRNet vs. HotCrab, respectively, for formation enthalpy (0.048 eV vs. 0.031 eV), band gap (0.047 eV vs. 0.048 eV), energy per atom (0.070 eV vs. 0.033 eV), and volume per atom (0.394 Å³ vs. 0.277 Å³).

We generate baseline RF metrics using a random forest regression with the Magpie CBFV as defined by Matminer²⁶. This is done using the scikit-learn Python package. The RF models were trained with $n_{\text{estimators}} = 500$ and default parameters.

4.4 PUBLICATION 2: CRABNET

DATA AVAILABILITY

Data is provided in its cleaned and pre-split form to ensure reproducible results, and with the hope that other researchers find it useful when benchmarking their own approaches. The processed data that is used in this study can be found on the GitHub repository⁵⁹ at the address <https://github.com/anthony-wang/CrabNet>. All raw data as well as scripts to process and split the datasets can be found in the GitHub repository⁵⁸ at the address https://github.com/kaaiian/mse_datasets.

CODE AVAILABILITY

We provide detailed instructions for the installation, training, and general usage of the open-source CrabNet on GitHub⁵⁹. In addition, pre-trained network weights for the CrabNet models reported in this work are available for download⁶⁴. The following files are available with this publication: (1) GitHub repository with the CrabNet source code, figures, and example property predictions: <https://github.com/anthony-wang/CrabNet>, (2) pre-trained weights for the CrabNet models reported in this work: <https://doi.org/10.5281/zenodo.4633866>, and (3) Supplementary Information. Finally, we recommend that interested readers consult the paper "Machine Learning for Materials Scientists: An Introductory Guide towards Best Practices"⁴ for a detailed treatment of best practices in machine learning and justification for many of the unmentioned experimental design decisions used in this work.

Received: 22 February 2020; Accepted: 26 April 2021;
Published online: 28 May 2021

REFERENCES

- Maier, W. F., Stöwe, K. & Sieg, S. Combinatorial and high-throughput materials science. *Angewandte Chemie (International ed. in English)* **46**, 6016–6067 (2007).
- Agrawal, A. & Choudhary, A. Perspective: materials informatics and big data: realization of the "fourth paradigm" of science in materials science. *APL Mater.* **4**, 053208 (2016).
- Barnard, A. S. Best practice leads to the best materials informatics. *Matter* **3**, 22–23 (2020).
- Wang, A. Y.-T. et al. Machine learning for materials scientists: an introductory guide toward best practices. *Chem. Mater.* **32**, 4954–4965 (2020).
- Jha, D. et al. ElemNet: deep learning the chemistry of materials from only elemental composition. *Sci. Rep.* **8**, 17593 (2018).
- Jha, D. et al. IRNet: a general purpose deep residual regression framework for materials discovery. In *Proc. 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining – KDD '19*, 2385–2393 (eds. Teredesai, A. et al.) (ACM Press, 2019).
- Xie, T. & Grossman, J. C. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Phys. Rev. Lett.* **120**, 145301 (2018).
- Schütt, K. T., Sauceda, H. E., Kindermans, P.-J., Tkatchenko, A. & Müller, K.-R. SchNet – A deep learning architecture for molecules and materials. *J. Chem. Phys.* **148**, 241722 (2018).
- Goodall, R. E. A. & Lee, A. A. Predicting materials properties without crystal structure: deep representation learning from stoichiometry. *Nat. Commun.* **11**, 6280 (2020).
- Ziletti, A., Kumar, D., Scheffler, M. & Ghiringhelli, L. M. Insightful classification of crystal structures using deep learning. *Nat. Commun.* **9**, 2775 (2018).
- Faber, F. A., Lindmaa, A., von Lilienfeld, O. A. & Armiento, R. Crystal structure representations for machine learning models of formation energies. *Int. J. Quantum Chem.* **115**, 1094–1101 (2015).
- Faber, F. A., Lindmaa, A., von Lilienfeld, O. A. & Armiento, R. Machine learning energies of 2 million elpasolite (ABC2D6) crystals. *Phys. Rev. Lett.* **117**, 135502 (2016).
- Kong, C. S. et al. Information-theoretic approach for the discovery of design rules for crystal chemistry. *J. Chem. Inform. Model.* **52**, 1812–1820 (2012).
- Fischer, C. C., Tibbetts, K. J., Morgan, D. & Ceder, G. Predicting crystal structure by merging data mining with quantum mechanics. *Nat. Mat.* **5**, 641–646 (2006).
- Curtarolo, S., Morgan, D., Persson, K. A., Rodgers, J. & Ceder, G. Predicting crystal structures with data mining of quantum calculations. *Phys. Rev. Lett.* **91**, 135503 (2003).
- Zhuo, Y., Mansouri Tehrani, A. & Brgoch, J. Predicting the band gaps of inorganic solids by machine learning. *J. Phys. Chem. Lett.* **9**, 1668–1673 (2018).
- Kauwe, S. K., Graser, J., Vazquez, A. & Sparks, T. D. Machine learning prediction of heat capacity for solid inorganics. *Integr. Mater. Manuf. Innov.* **7**, 43–51 (2018).

18. Oliynyk, A. O. et al. High-throughput machine-learning-driven synthesis of full-heusler compounds. *Chem. Mater.* **28**, 7324–7331 (2016).
19. Hautier, G., Fischer, C. C., Jain, A., Mueller, T. & Ceder, G. Finding nature's missing ternary oxide compounds using machine learning and density functional theory. *Chem. Mater.* **22**, 3762–3767 (2010).
20. Mansouri Tehrani, A. et al. Machine learning directed search for ultraincompressible, superhard materials. *J. Am. Chem. Soc.* **140**, 9844–9853 (2018).
21. Graser, J., Kauwe, S. K. & Sparks, T. D. Machine learning and energy minimization approaches for crystal structure predictions: a review and new horizons. *Chem. Mater.* **30**, 3601–3612 (2018).
22. Choudhary, K., DeCost, B. & Tavazza, F. Machine learning with force-field-inspired descriptors for materials: fast screening and mapping energy landscape. *Phys. Rev. Mater.* **2**, 083801 (2018).
23. Kauwe, S. K., Graser, J., Murdock, R. J. & Sparks, T. D. Can machine learning find extraordinary materials? *Comput. Mater. Sci.* **174**, 109498 (2020).
24. Gaultois, M. W. et al. Perspective: web-based machine learning models for real-time screening of thermoelectric materials properties. *APL Mater.* **4**, 053213 (2016).
25. de Jong, M. et al. A statistical learning framework for materials science: application to elastic moduli of k-nary inorganic polycrystalline compounds. *Sci. Rep.* **6**, 34256 (2016).
26. Glaudell, A. M., Cochran, J. E., Patel, S. N. & Chabinyk, M. L. Impact of the doping method on conductivity and thermopower in semiconducting polythiophenes. *Adv. Energy Mater.* **5**, 1401072 (2015).
27. Zhang, S. B. The microscopic origin of the doping limits in semiconductors and wide-gap materials and recent developments in overcoming these limits: a review. *J. Phys.: Condensed Matter* **14**, R881–R903 (2002).
28. Sheng, L., Wang, L., Xi, T., Zheng, Y. & Ye, H. Microstructure, precipitates and compressive properties of various holmium doped NiAl/Cr(Mo,Hf) eutectic alloys. *Mater. Design* **32**, 4810–4817 (2011).
29. Mansouri Tehrani, A. et al. Atomic substitution to balance hardness, ductility, and sustainability in molybdenum tungsten borocarbide. *Chem. Mater.* **31**, 7696–7703 (2019).
30. Mihailovich, R. E. & Parpia, J. M. Low temperature mechanical properties of boron-doped silicon. *Phys. Rev. Lett.* **68**, 3052–3055 (1992).
31. Qu, Z., Sparks, T. D., Pan, W. & Clarke, D. R. Thermal conductivity of the gadolinium calcium silicate apatites: effect of different point defect types. *Acta Materialia* **59**, 3841–3850 (2011).
32. Sparks, T. D., Fuierer, P. A. & Clarke, D. R. Anisotropic thermal diffusivity and conductivity of La-doped strontium niobate SrNb₂O₇. *J. Am. Ceramic Soc.* **93**, 1136–1141 (2010).
33. Grimvall, G. *Thermophysical Properties of Materials* 1st edn. (North Holland, Amsterdam, 1999).
34. Gaumé, R., Viana, B., Vivien, D., Roger, J.-P. & Fournier, D. A simple model for the prediction of thermal conductivity in pure and doped insulating crystals. *Appl. Phys. Lett.* **83**, 1355–1357 (2003).
35. Murdock, R. J., Kauwe, S. K., Wang, A. Y.-T. & Sparks, T. D. Is domain knowledge necessary for machine learning materials properties? *Integr. Mater. Manuf. Innov.* **9**, 221–227 (2020).
36. Dunn, A., Wang, Q., Ganose, A., Dopp, D. & Jain, A. Benchmarking materials property prediction methods: the Matbench test set and Automaterminer reference algorithm. *npj Comput. Mater.* **6**, 138 (2020).
37. Vaswani, A. et al. in *Advances in Neural Information Processing Systems* (eds. Guyon, I. et al.) (Curran Associates Inc., 2017).
38. Tang, G., Müller, M., Rios, A. & Sennrich, R. Why self-attention? A targeted evaluation of neural machine translation architectures. In *Proc. 2018 Conference on Empirical Methods in Natural Language Processing* (eds. Riloff, E. et al.) 4263–4272 (Association for Computational Linguistics, 2018).
39. Al-Rfou, R., Choe, D., Constant, N., Guo, M. & Jones, L. Character-level language modeling with deeper self-attention. *Proc. AAAI Conf. Artificial Intelligence* **33**, 3159–3166 (2019).
40. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. 2019 Conference of the North American Chapter of the Association for Computational Linguistics* (eds. Burstein, J., Doran, C. & Solorio, T.) 4171–4186 (Association for Computational Linguistics, 2019).
41. Yu, A. W. et al. QANet: Combining local convolution with global self-attention for reading comprehension. In *Proc. International Conference on Learning Representations (ICLR)* (2018).
42. Yang, Z. et al. XLNet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems* (eds. Wallach, H. M. et al.) (Curran Associates Inc., 2019).
43. Huang, C.-Z. A. et al. Music transformer. In *Proc. International Conference on Learning Representations (ICLR)* (2019).
44. Zhang, H., Goodfellow, I., Metaxas, D. & Odena, A. Self-attention generative adversarial networks. In *Proc. 36th International Conference on Machine Learning (ICML)* (eds. Chaudhuri, K. & Salakhutdinov, R.) 7354–7363 (PMLR, 2019).
45. Dai, T., Cai, J., Zhang, Y., Xia, S.-T. & Zhang, L. Second-order attention network for single image super-resolution. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (eds. CVPR Editors) 11057–11066 (IEEE, 2019).
46. Zhang, Y. et al. Image super-resolution using very deep residual channel attention networks. In *Computer Vision – ECCV 2018* (eds. Ferrari, V. et al.) vol. 11211, 294–310 (Springer International Publishing, 2018).
47. Zhang, Y., Li, K., Li, K., Zhong, B. & Fu, Y. Residual non-local attention networks for image restoration. In *Proc. International Conference on Learning Representations (ICLR)* (2019).
48. Kim, T. H., Sajjadi, M. S. M., Hirsch, M. & Schölkopf, B. Spatio-temporal transformer network for video restoration. In *Computer Vision – ECCV 2018* (eds. Ferrari, V. et al.) vol. 11207, 111–127 (Springer International Publishing, 2018).
49. Wang, X., Chan, K. C. K., Yu, K., Dong, C. & Loy, C. C. EDVR: video restoration with enhanced deformable convolutional networks. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 1954–1963 (IEEE, 2019).
50. Vinyals, O. et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* **575**, 350–354 (2019).
51. Baker, B. et al. Emergent tool use from multi-agent autocurricula. In *Proc. International Conference on Learning Representations (ICLR)* (2020).
52. Zheng, S., Yan, X., Yang, Y. & Xu, J. Identifying structure-property relationships through SMILES syntax analysis with self-attention mechanism. *J. Chem. Inform. Model.* **59**, 914–923 (2019).
53. Schwaller, P. et al. Molecular transformer: a model for uncertainty-calibrated chemical reaction prediction. *ACS Central Sci.* **5**, 1572–1583 (2019).
54. Clement, C. L., Kauwe, S. K. & Sparks, T. D. Benchmark AFLOW data sets for machine learning. *Integr. Mater. Manuf. Innov.* **9**, 153–156 (2020).
55. Bartel, C. J. et al. A critical examination of compound stability predictions from machine-learned formation energies. *npj Comput. Mater.* **6**, 97 (2020).
56. Kirkin, S. et al. The Open Quantum Materials Database (OQMD): assessing the accuracy of DFT formation energies. *npj Comput. Mater.* **1**, 15010 (2015).
57. Ward, L. et al. Matminer: an open source toolkit for materials data mining. *Comput. Mater. Sci.* **152**, 60–69 (2018).
58. Kauwe, S. K. Online GitHub repository for mse_datasets. https://github.com/kaaiian/mse_datasets (2020).
59. Wang, A. Y.-T. & Kauwe, S. K. Online GitHub repository for the paper "Compositionally-Restricted Attention-Based Network for Materials Property Prediction". <https://github.com/anthony-wang/CrabNet> (2020).
60. Tshitoyan, V. et al. Unsupervised word embeddings capture latent knowledge from materials science literature. *Nature* **571**, 95–98 (2019).
61. Ward, L., Agrawal, A., Choudhary, A. & Wolverton, C. A general-purpose machine learning framework for predicting properties of inorganic materials. *npj Comput. Mater.* **2**, 16028 (2016).
62. Zhang, M. R., Lucas, J., Hinton, G. & Ba, J. in *Advances in Neural Information Processing Systems* (eds. Wallach, H. M. et al.) (Curran Associates Inc., 2019).
63. You, Y. et al. Large batch optimization for deep learning: training BERT in 76 minutes. In *Proc. International Conference on Learning Representations (ICLR)* (2020).
64. Wang, A. Y.-T., Kauwe, S. K., Murdock, R. J. & Sparks, T. D. Trained network weights for the paper "Compositionally-Restricted Attention-Based Network (CrabNet)". <https://doi.org/10.5281/zenodo.4633866> (2021).
65. Castelli, I. E. et al. Computational screening of perovskite metal oxides for optimal solar light capture. *Energy Environ. Sci.* **5**, 5814–5819 (2012).
66. Jain, A. et al. Commentary: The Materials Project: a materials genome approach to accelerating materials innovation. *APL Mater.* **1**, 011002 (2013).
67. Ong, S. P. et al. The Materials Application Programming Interface (API): a simple, flexible and efficient API for materials data based on REpresentational State Transfer (REST) principles. *Comput. Mater. Sci.* **97**, 209–215 (2015).
68. Petousis, I. et al. High-throughput screening of inorganic compounds for the discovery of novel dielectric and optical materials. *Sci. Data* **4**, 160134 (2017).
69. de Jong, M. et al. Charting the complete elastic properties of inorganic crystalline compounds. *Sci. Data* **2**, 150009 (2015).
70. National Institute of Standards and Technology (NIST). NIST JARVIS-DFT Database. <https://www.nist.gov/programs-projects/jarvis-dft> (2017).
71. Petretto, G. et al. High-throughput density-functional perturbation theory phonons for inorganic materials. *Sci. Data* **5**, 180065 (2018).
72. Conduit, G. & Bajaj, S. Mechanical properties of some steels: ID: 153092 - Version 3 <https://citrine.com/datasets/153092/> (2017).

ACKNOWLEDGEMENTS

The authors gratefully acknowledge support from the NSF CAREER Award DMR 1651668. The authors also thank the Berlin International Graduate School in Model

and Simulation-based Research as well as the German Academic Exchange Service (program number 57438025) for their financial support. Special thanks are given to Dr. Aleksander Gurlo and Dr. Mathias Czasny for advising and supporting Anthony Yu-Tung Wang and for encouraging his collaborative stay at the University of Utah. The authors thank the creators of AFLOW and Materials Project for the creation of the databases and for making the material properties available for this study. In addition, the authors express their gratitude to the open-source software community, for developing the excellent tools used in this research, including but not limited to Python, Pandas, NumPy, matplotlib, scikit-learn, and PyTorch. Last but not least, the authors thank OpenAI, the researchers at Hugging Face, and Adam King for their contribution to [TalkToTransformer.com](https://github.com/huggingface/transformers). The underlying Transformer-powered GPT-2 model was used to generate text for the closing lines of this publication.

AUTHOR CONTRIBUTIONS

A.Y.T.W. and S.K.K. jointly and in equal amounts conceived, developed the concept, and implemented the algorithms, code and visualizations described in this work. A.Y.T.W. and S.K.K. jointly analyzed the results. R.J.M. assisted with developing the architecture and provided insight and guidance during model optimization and training. All authors discussed the results and contributed to the writing of the manuscript.

COMPETING INTERESTS

The authors declare no competing interests.

ADDITIONAL INFORMATION

Supplementary information The online version contains supplementary material available at <https://doi.org/10.1038/s41524-021-00545-1>.

Correspondence and requests for materials should be addressed to T.D.S.

Reprints and permission information is available at <http://www.nature.com/reprints>

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

© The Author(s) 2021

Supplementary Information for the article “Compositionally-Restricted Attention-Based Network for Materials Property Predictions”

Authors: Anthony Yu-Tung Wang, Steven K. Kauwe, Ryan J. Murdock, and Taylor D. Sparks

URL of CrabNet GitHub: <https://github.com/anthony-wang/CrabNet>

DOI of trained CrabNet model weights: <https://doi.org/10.5281/zenodo.4633866>

Supplementary Methods

As noted in the publication, we have taken additional steps to ensure that (1) no duplicate compositions were present in each of the train, validation, and test datasets, and that (2) if a composition exists in the train or validation dataset, all compounds with the same composition are removed from the validation and test datasets. Below are the details regarding the dataset processing steps.

First, all structural information (if any) was removed from the original datasets.

Then, the pre-split datasets for each material property were examined for any duplicate compositions in each of the train, validation and test splits. In our case, the duplicate compositions come from the fact that we have removed structural information from the datasets.

Thus, compounds which may exist with multiple crystal structures (with different properties) are now recorded in the dataset as the same composition, having different target values. We remove duplicate target values of the same composition by taking the mean of all available target values.

Differently, for the OQMD and MP datasets, we first sorted compounds which had duplicate entries by order of increasing formation enthalpy. Then, we took the property values associated with the lowest formation enthalpy value for each compound and used those as the target values for the compounds in each of the OQMD and MP datasets.

Second, we examine, for each of the material properties, the compositions present within the train, validation, and test splits. If a composition exists in the train dataset and is found in the validation or test dataset, we remove that composition in the validation or test dataset.

Likewise, we do the same for compositions in the validation dataset being found in the test dataset. This ensures that there is no leakage of chemical information between the dataset splits. Here we note that we use the normalized composition to detect matching composition, Na_2Cl_2 is treated as being the same composition as NaCl . This is comparable to the way how atomic fractional amounts are featurized by the CrabNet EDM featurization scheme (discussed in the publication).

We note additionally that, in order to remain comparable with other works which use the datasets including the structural information (such as the Automatminer publication), we have elected not to re-split the dataset splits into different train/validation/test datasets.

Instead, we have taken the original datasets from the publications and applied the data processing steps as mentioned above, to retain as much of the original dataset splits as possible.

4 NOVEL ATTENTION-BASED LEARNING OF MATERIALS PROPERTIES

CrabNet is composed of the following PyTorch modules. These are used to build the CrabNet architecture as specified in the files `crabnet/{kingcrab|model}.py`.

Model architecture: `out_dims, d_model, N, heads`
`3, 512, 3, 4`

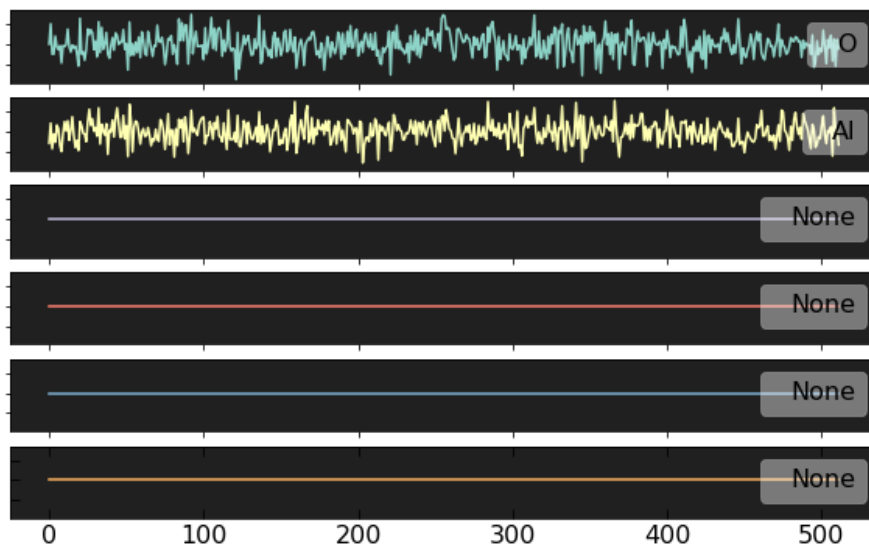
```
CrabNet(
  (encoder): Encoder(
    (embed): Embedder(
      (fc_mat2vec): Linear(in_features=200, out_features=512, bias=True)
      (cbfv): Embedding(119, 200)
    )
    (pe): FractionalEncoder()
    (ple): FractionalEncoder()
    (transformer_encoder): TransformerEncoder(
      (layers): ModuleList(
        (0): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): _LinearWithBias(in_features=512,
out_features=512, bias=True)
          )
          (linear1): Linear(in_features=512, out_features=2048,
bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=2048, out_features=512,
bias=True)
          (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0.1, inplace=False)
          (dropout2): Dropout(p=0.1, inplace=False)
        )
        (1): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): _LinearWithBias(in_features=512,
out_features=512, bias=True)
          )
          (linear1): Linear(in_features=512, out_features=2048,
bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=2048, out_features=512,
bias=True)
          (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0.1, inplace=False)
          (dropout2): Dropout(p=0.1, inplace=False)
        )
        (2): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): _LinearWithBias(in_features=512,
out_features=512, bias=True)
```

4.4 PUBLICATION 2: CRABNET – SI

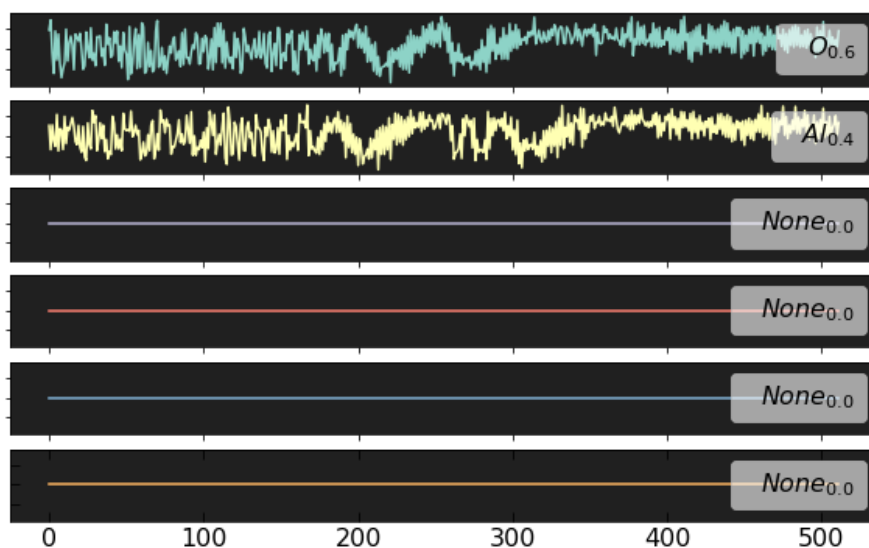
```
)
  (linear1): Linear(in_features=512, out_features=2048,
bias=True)
  (dropout): Dropout(p=0.1, inplace=False)
  (linear2): Linear(in_features=2048, out_features=512,
bias=True)
  (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
  (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
  (dropout1): Dropout(p=0.1, inplace=False)
  (dropout2): Dropout(p=0.1, inplace=False)
)
)
)
)
(output_nn): ResidualNetwork
)
```

Number of params: 11987206

Supplementary Figures

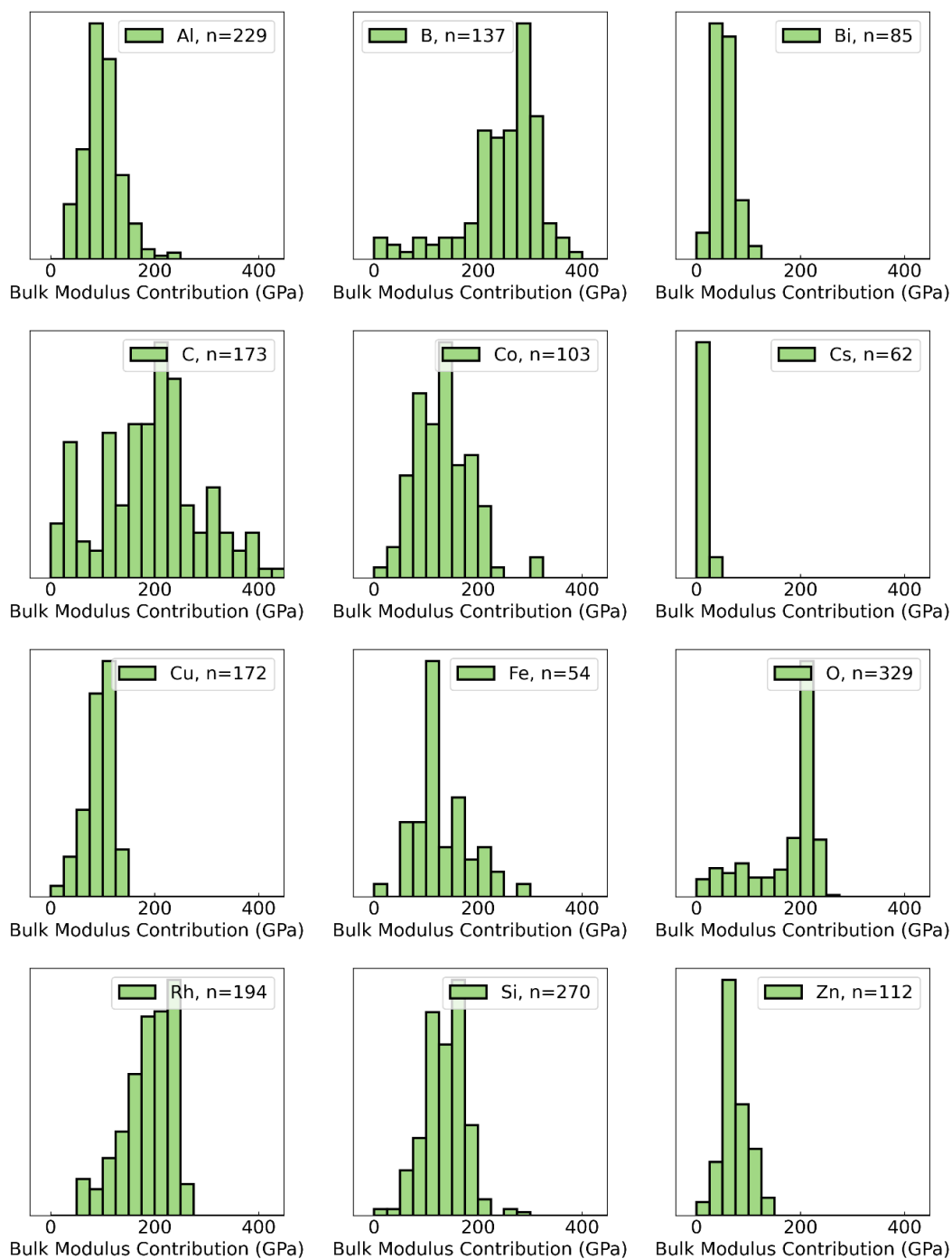


Supplementary Figure 1. Visualization of an example EDM embedding of the compound Al_2O_3 . The elements shown were featurized using the “mat2vec” element properties. Note that the fractional embedding has not been added to the feature vectors of O and Al yet. “None” refers to no element present in the EDM.



Supplementary Figure 2. Visualization of an example EDM embedding of the compound Al_2O_3 , with the fractional embedding added element-wise to the element property embeddings. “None” refers to no element present in the EDM.

4.4 PUBLICATION 2: CRABNET – SI



Supplementary Figure 3. Selected additional examples of the per-element contribution plots (Figure 3b in the publication) from the CrabNet model trained using the AFLOW bulk modulus dataset. See the directory figures/contributions in the CrabNet repository for all of the plots.

5 Interpretable deep learning with CrabNet

5.1 Lack of interpretable modeling in materials science

Despite the excellent performance and adaptability of deep learning (DL) models for materials informatics (MI) as seen in recent academic publications, the adoption of these models in the industry is comparatively limited. Arguably, the major reason for this discrepancy between the popularity of DL in academic research and the lack of its adoption in industry is the widespread use of black-box models which are not directly interpretable by end users. This discrepancy is termed the “interpretability-adoption gap” and is one of the motivations for the development of explainable AI (XAI) methods.

Although there are established mathematical and statistical methods for evaluating model performance, it is nonetheless often desirable for users and decision-makers to have the model metrics supplemented with an intuitive understanding of the modeling and decision-making process. This is especially helpful for end users who are not well-versed in the complexities of DL, model evaluation, and statistics.

In order to further increase the adoption of DL models in MI, it is not enough to simply provide well-documented and open-source code with an accompanying publication. Efforts should also be made to increase the interpretability of the models, either by building in methods to inspect the modeling process in an existing model (“post-hoc interpretability”) or by designing and building models with interpretability as a main goal (“intrinsic interpretability”). This will not only lead to further understanding of the model decision-making processes and additional chemical insights, but also promote the adoption of DL modeling and MI techniques in academia and industry alike.

These requirements are achieved in this work by incorporating intrinsic model interpretability in CrabNet using several methods aimed at extracting additional information out of the modeling process. The architecture and EDM encoding scheme of CrabNet,

when combined with additional model interpretability and inspection methods as introduced in this work, lead to several important advantages:

- 1) Since the element identity is preserved in the input EDM data throughout the modeling process, it becomes possible to track the element representations and how they are modified by CrabNet as they pass through the model architecture. This enables the examination of the changing element behavior as well as the per-element contributions to the material property throughout modeling.
- 2) The element vectors can be extracted from CrabNet and examined to explore the learned element trends and relationships (such as element identity, similarity and abundance) on a per-property basis. This can be extended to the compound level to explore clustering and relationships between compounds in a given dataset.
- 3) The self-attention mechanism can be used to explicitly encode element-element interactions within a compound, allowing CrabNet to capture both the entire representation of a chemical compound while still being sensitive to interactions between specific elements. These element-element interactions can be visualized to gain additional insights about the modeling process and chemical behaviour of the elements.
- 4) The EDM representation of elements and compounds as well as the attention mechanism contain an incredible amount of model-internal information and offer an extraordinary opportunity to increase the interpretability of CrabNet models. Further examination and visualization of this information may bring about a better understanding of the modeling process and additional insights about the chemical phenomena governing materials behaviours. This may lead to increased confidence in CrabNet and ultimately accelerate the adoption of CrabNet and other data-driven methods in materials science.

The third publication resulting from this thesis work aims to address these issues through the improvement of model interpretability in CrabNet. The publication and its accompanying supplementary information (SI) are inserted in the following pages.

5.2 Publication 3: CrabNet for Explainable Deep Learning in Materials Science: Bridging the Gap Between Academia and Industry

Title	CrabNet for Explainable Deep Learning in Materials Science: Bridging the Gap Between Academia and Industry
Authors	A. Y.-T. Wang, M. S. Mahmoud, M. Czasny, and A. Gurlo
Journal	Integrating Materials and Manufacturing Innovation
Publisher	Springer International Publishing AG
Publication date	January 17, 2022
Reference	<i>Integrating Materials and Manufacturing Innovation</i> , 2022
DOI	10.1007/s40192-021-00247-y
Supporting information	The SI can be downloaded from: https://doi.org/10.1007/s40192-021-00247-y
My contribution	Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data Curation, Visualization, Writing – Original Draft, Writing – Review & Editing (together with M.S.M), Project administration.

The article is inserted in the following pages. The article is reprinted under the open access CC BY license (<https://creativecommons.org/licenses/by/4.0/>).



CrabNet for Explainable Deep Learning in Materials Science: Bridging the Gap Between Academia and Industry

Anthony Yu-Tung Wang¹ · Mahamad Salah Mahmoud² · Mathias Czasny¹ · Aleksander Gurlo¹Received: 22 October 2021 / Accepted: 5 December 2021
© The Author(s) 2022

Abstract

Despite recent breakthroughs in deep learning for materials informatics, there exists a disparity between their popularity in academic research and their limited adoption in the industry. A significant contributor to this “interpretability-adoption gap” is the prevalence of black-box models and the lack of built-in methods for model interpretation. While established methods for evaluating model performance exist, an intuitive understanding of the modeling and decision-making processes in models is nonetheless desired in many cases. In this work, we demonstrate several ways of incorporating model interpretability to the structure-agnostic Compositionally Restricted Attention-Based network, CrabNet. We show that CrabNet learns meaningful, material property-specific element representations based solely on the data with no additional supervision. These element representations can then be used to explore element identity, similarity, behavior, and interactions within different chemical environments. Chemical compounds can also be uniquely represented and examined to reveal clear structures and trends within the chemical space. Additionally, visualizations of the attention mechanism can be used in conjunction to further understand the modeling process, identify potential modeling or dataset errors, and hint at further chemical insights leading to a better understanding of the phenomena governing material properties. We feel confident that the interpretability methods introduced in this work for CrabNet will be of keen interest to materials informatics researchers as well as industrial practitioners alike.

Keywords Materials informatics · Deep learning · Self-attention · Interpretability · Explainable AI · XAI

Introduction

Machine learning (ML) in materials informatics (MI) has received significant attention in the academic research world and is gaining widespread adoption [1–5]. More specifically, it has recently been extensively studied for its use in the research and design of novel inorganic materials [6–10]. This is enabled by three major developments: (1) the increasing number of material property datasets as well as the improvement in dataset quality and variety, (2) the rapid pace and development of new ML models tailored to

addressing different challenges in materials science (*e.g.*, regression, classification), supplemented by (3) the increase in available computing power and accessibility to ML and deep learning tools. The combination of these developments led to improved capabilities in the exploration and modeling of material properties in the academic world.

Classical ML methods (*e.g.*, linear regression, random forest, support vector machines) have successfully been used for the regression and classification of many material properties [11–17]. These methods usually rely on the featurization of the input chemical formulae into numerical features that are usable by the models. Typically, this is achieved through the use of a composition-based feature vector (CBFV), which uses descriptive statistics of the properties of constituent atoms in each compound to uniquely represent it [18]. Some common CBFV feature sets are Oliynyk, Magpie, Jarvis and mat2vec [11, 12, 19, 20]. Here, a distinction is made between physically derived CBFVs (with features based on measurable element properties) like Oliynyk, Magpie and Jarvis, and

✉ Anthony Yu-Tung Wang
anthony.wang@ceramics.tu-berlin.de

¹ Technische Universität Berlin, Institute of Material Science and Technology, Fachgebiet Keramische Werkstoffe/Chair of Advanced Ceramic Materials, Straße des 17. Juni 135, 10623 Berlin, Germany

² Department of Computer and Data Sciences, Case Western Reserve University, Cleveland, OH 44106, USA

computationally derived CBFVs (with features obtained from computational or deep learning models) like `mat2vec`. For some properties, additional features such as structural information and processing or measurement conditions are included to further improve model performance [2, 16, 21, 22].

In more recent years, deep learning (DL) models have gained widespread popularity in MI due to numerous advantages compared to classical ML methods. Some examples are ElemNet, CGCNN, MEGNet, DimeNet++, and ALIGNN [23–27]. More recently, graph neural network (GNN) models incorporating attention-based mechanisms such as CrabNet, Roost and H-CLMP have gained increasing popularity [28–30]. GNNs have shown improved performance compared to other DL models, particularly in the absence of structural information as model inputs. Another advantage of GNNs is that the inductive biases built into the model and the input data structure are more suited to the learning of material properties, since the interactions between the atoms in the compound can be modeled as weighted interactions between nodes in a graph. In CrabNet, for example, the atom representations are either based on a CBFV feature (`mat2vec` element vectors) or a non-CBFV feature (onehot element vectors) [28]. For the sake of clarity, the remaining text will use the acronym DL to refer to both deep learning (DL) and graph neural network (GNN) models and methods.

Unfortunately, while DL methods show superb performance in modeling material properties, the element features used by these models typically do not represent any measurable physical property of the elements themselves. Instead, the element representations are learned from the data during the model training process. Therefore, they do not directly provide useful information or insights that can be interpreted by humans. This is different from the CBFV representation typically used in classical ML, where the features represent properties of the elements which are known *a priori*, such as the atomic mass, first ionization energy, or number of valence electrons.

Despite the high performance of the DL models, there is a disparity between their extensive study in academic research and their limited adoption in the industry for the exploration of materials. We term this disparity the “interpretability-adoption gap”. One significant hurdle to the widespread adoption of the often “black-box” models is the lack of built-in methods for model interpretation. While there are established methods of evaluating model performance in academia [14, 31–33], those who are less familiar with DL typically require more intuition into how the models function before they can fully trust the results. Particularly in industry, where there is usually a lower risk tolerance compared to academia, findings based on black-box models and vague model evaluation criteria are not enough to justify making high-stakes decisions such as investing in new research [5,

34–38]. Tangible methods of investigating and understanding model decision-making processes are therefore required to facilitate their adoption in an industrial setting [39].

This led to the development of explainable AI (XAI), which aims to introduce methods for deciphering the internal workings of black-box models and thus enabling users to understand the modeling processes and results [39, 40]. Examples of XAI in research fields outside of MI include: visualizing word embeddings in natural language processing [41–43], inspecting decision-making processes in reinforcement learning [44–46], visualizing pixel importances [47, 48], or segmenting in computer vision [49, 50]. To date, however, XAI techniques have—with the exception of a few works employing classical ML—largely been underexplored for DL in the MI field [10, 51, 52].

Two common post-hoc model-agnostic methods for obtaining explainable models in classical ML are SHAP and LIME [39, 53–55]. Both of these methods are built on top of existing black-box models and use local feature perturbation to estimate the contributions from input features towards the predictions. Other models such as random forest, gradient boosting, and lasso regression inherently provide model interpretability via the use of internal feature importance metrics and (in some models) through bootstrap sampling and feature sampling [39, 51, 56]. Nonetheless, these techniques require that the individual features of the input data are meaningful and represent a measurable feature or physical property. This works in the domain of classical ML and when using a physically derived CBFV to featurize compounds; however, this is not the case for DL methods where the features typically do not reflect measurable values. Thus, these traditional ways of model interpretability fall short in use for the DL models.

Therefore, it is the goal of this work to explore how to increase model interpretability in DL models specifically for applications in MI. Here, we demonstrate how parts of the typically black-box modeling process can be communicated visually and in an interpretable way, using our attention-based model, CrabNet [28]. We have extended CrabNet’s architecture to enable intrinsic interpretability using several methods to be discussed below. In this regard, we lay the first bricks in the bridge spanning the interpretability-adoption gap between academia and industry. This will not only aid researchers in further developing complex models with interpretability in focus, but also promote the adoption of these modeling methods in the materials science industry.

Results and Discussion

The results of this study are described in five subsections. We first compare the element embeddings learned by CrabNet against other CBFV feature sets from the literature,

5 INTERPRETABLE DEEP LEARNING WITH CRABNET

Integrating Materials and Manufacturing Innovation

and show how chemical behavior and patterns in element properties can be learned entirely from the training data for each material property. We also show that the learned element representations are comparable to physically derived CBFVs. Secondly, as part of this analysis, we characterize the element prevalence imbalance in the datasets using the Shannon equitability index and relate that to the quality of the learned element embeddings. Third, we further examine how the element representations are successively updated using information about their chemical environment in the compounds, and how they may be used to gain additional insights about element behaviors in different environments. Fourth, we inspect how entire chemical compounds can also be adequately captured using the EDMs and subsequently visualized. We identify interesting trends in the compound representations relating the bond character and number of elements in the compounds to the material property and prediction error, and discuss how such visualizations can lead to additional understanding about the modeling process and the underlying materials chemistry. Lastly, we explore how the self-attention mechanism in CrabNet can be visualized in the form of videos and used to further examine the modeling process, leading to potential new insights about the chemical interactions within a compound. While we use the OQMD_Bandgap dataset to demonstrate the analyses, we note that similar analyses can be also carried out with any of the 28 materials datasets presented in this work.

Learning Meaningful and Per-Property Element Representations

Element representations were obtained as featurized CBFVs, which are fixed-length vectors where each element is uniquely described by the same set of features [12, 18].

For the Oliynyk, Magpie and mat2vec element property feature sets, we use the published vectors to represent the elements [18, 20]. For the CrabNet element representations, we extract the element vectors from the element-derived matrices (EDMs) at the output of the embedding layer (please refer to the CrabNet publication for architecture details [28]). We then examine the similarity between two element vectors x and y by computing the Pearson correlation coefficient r using Equation 1:

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 (y_i - \bar{y})^2}} \quad (1)$$

where n is the number of features, x_i and y_i are the values of the i th feature, and \bar{x} and \bar{y} are the mean values of x and y , respectively.

The correlation r ranges from -1 to 1; the higher or lower the value of r is, the more correlated or anticorrelated are the features that describe the elements, respectively. A value of zero means that there are no correlations between the features of the elements. We compute the pairwise correlation coefficients between the element vectors for all elements and for all element property representations, and show these as heatmaps in Fig. 1. Note that the plots are cropped to the range of elements of the Oliynyk heatmap to aid comparison; please refer to supplementary Fig. S-1 in the supplementary information (SI) for the full heatmaps. In addition, interactive versions of the plots are provided in the SI.

Here, we can observe that element vectors based on the Oliynyk and Magpie CBFVs contain large regions of similar color in the heatmap. The regions of similar color indicate that the element representations are either highly correlated or highly anticorrelated with each other.

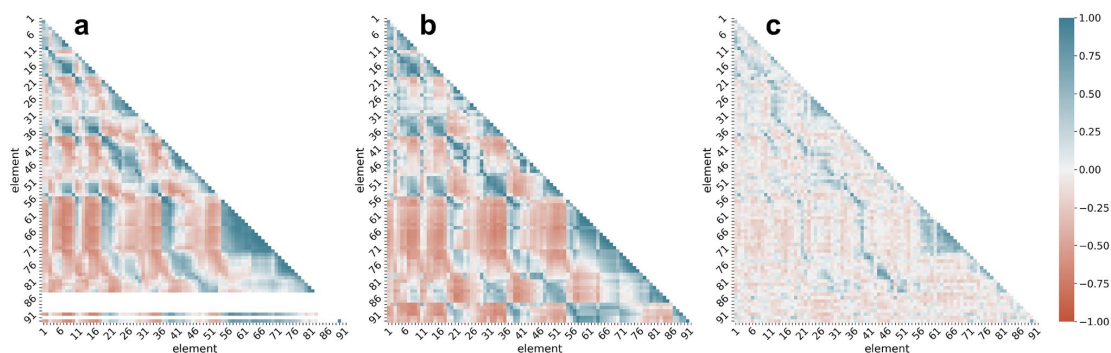


Fig. 1 Heatmaps of Pearson correlation matrices between element vectors featurized using **a** Oliynyk, **b** Magpie, and **c** mat2vec element property feature sets. The x - and y -axes are labeled with the atomic numbers. Each cell at coordinate (x, y) represents the correlation between the corresponding elements with atomic numbers x and y .

Blue represents a high correlation and red represents a high anticorrelation. For the interest of comparison, the heatmaps are truncated to the dimensions of the Oliynyk heatmap. Empty rows indicate that no element vector is available

Furthermore, these regions are very similar between the two CBFVs. This is expected, since the CBFV features are based on physical properties of the elements. Thus, elements with similar physical properties will be more correlated while dissimilar elements will be more anticorrelated. Accordingly, the large colored regions typically correspond to similarities and dissimilarities between elements from families in the periodic table, such as alkali metals, alkaline earth metals, transition metals, metalloids and reactive nonmetals.

On the other hand, the element vectors from a DL model such as `mat2vec` do not exhibit such prominent behavior. Overall, the elements show less correlation with each other, and—with the exception of a few areas (to be discussed in later sections)—do not show large continuous regions of similar color. This is due to the fact that the starting element representations in DL models are randomly initialized and are not based on physical properties of the elements. These vector representations of the elements are only updated by the model throughout the training process using the training data. Thus, the correlation patterns that can be observed in this figure represent distinct patterns that the DL model has learned solely from the provided data.

We also note that a different number of element vectors are recorded in the feature sets. For the Oliynyk and Magpie CBFVs, only the elements up to uranium and berkelium are reported, respectively, while vectors up to the element oganesson are provided by `mat2vec` (please refer to supplementary Fig. S-1 in the SI for the uncropped heatmaps). Particularly for the Oliynyk CBFV, some element vectors are missing, as visible by the empty rows in the heatmap. This disparity in the availability of element vectors between different CBFVs can be caused by reasons such as the instability or rarity of elements, lack of adequate information about the elements, or the inability to measure properties about the elements. The lack of element vectors in some material property feature sets can limit their applicability for certain tasks (such as when studying rare elements) and will be discussed in more detail in later sections.

In addition to learning element representations for a general purpose in materials science, such in the case of `mat2vec`, DL methods can also learn to relate element characteristics on a material property-specific basis. For example, element embeddings were extracted from the CrabNet and HotCrab models which were reproduced using the supplied model weights and the source code [57, 58]. The CrabNet and HotCrab models use `mat2vec` and onehot-encoded element features as the starting element representations, respectively. These features are then fine-tuned by the models for each of the 28 reported datasets. We extract one set of element embeddings from each layer of the models. Then, the Pearson correlation between the element vectors are calculated and shown in Fig. 2.

In this work, we use the OQMD_Bandgap dataset to demonstrate our findings. Additional example plots for other properties can be found in the SI. The OQMD datasets are widely used by researchers to evaluate model performance. For detailed information about the OQMD_Bandgap dataset as well as information and discussion about the calculated values, please see the literature [59–61].

Here, we can observe that both CrabNet and HotCrab are able to learn embeddings for each element of the periodic table, and that the correlations between the elements have a similar pattern, irrespective of the starting element representation (`mat2vec` or onehot). The observed correlation patterns are also similar to the `mat2vec` patterns as seen in Fig. 1c. The ability of both CrabNet and HotCrab models to learn similar element embeddings despite having drastically different starting representations is encouraging, and further suggests that domain knowledge is not necessarily required for element featurization if a sufficient quantity and quality of training data is available [18]. This finding is corroborated by the similarly good performance of both models across a wide range of material properties [28]. Interestingly, for deeper layers of the models (Fig. 2b and d), more intense correlation patterns between the elements emerge. This is likely attributed to the self-attention-based learning mechanism of the underlying CrabNet models. At each successive layer within the model, information about additional element-element interactions within the compound (*i.e.*, the chemical environment) are successively taken into account when updating the identity of an element within that compound. As a result, the deeper the layer within the model, the more complex the element interactions—and the element representations—become.

It is also interesting to note the diagonal and horizontal patterns which can be observed in all of the correlation matrices. For example, in Fig. 2d there is a 45-degree diagonal, blue line that can be seen in the correlation matrix starting at the coordinates (13, 31) (corresponding to the element pair (Al, Ga)) and continuing until (40, 58) (corresponding to (Zr, Ce)). This line highlights the well-known periodic law which states that elements with similar chemical properties fall into recurring periodic groups. Please refer to supplementary Fig. S-2 for the enlarged version of the annotated heatmap and for correlation plots for other material properties. Another observation is the triangular region of high correlation between (57, 57) and (71, 71), which indicates that the first-row elements of the *f*-block are highly similar to each other. A similar triangular region can be observed between (23, 23) and (29, 29), indicating similarities between some first-row elements of the *d*-block. Lastly, the vertical blue line starting at the coordinates (39, 57) and continuing to (39, 71) indicate the chemical similarities between yttrium and the first-row elements of the *f*-block. These and other patterns can also be observed in

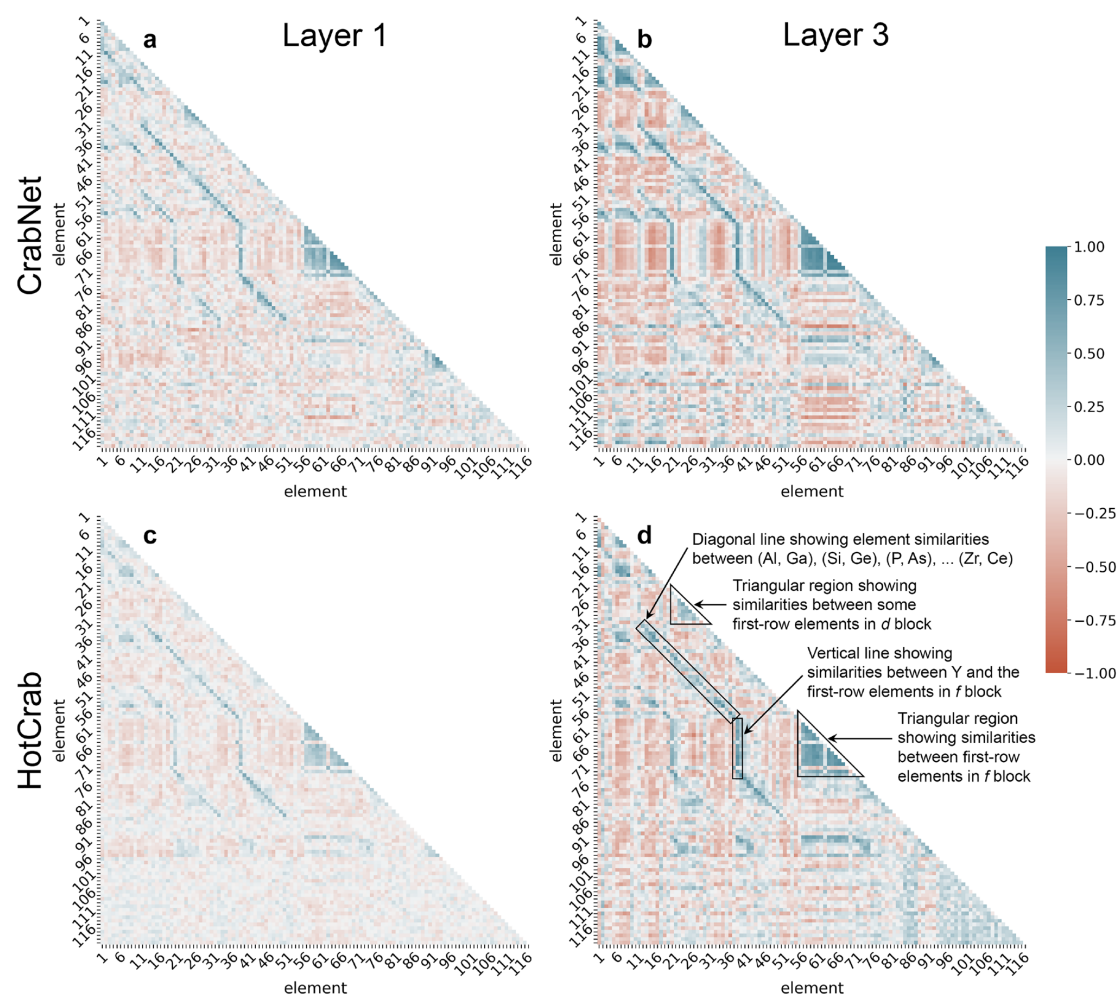


Fig. 2 Heatmaps of Pearson correlation matrices between element vectors extracted from CrabNet and HotCrab. These element representations are learned entirely from data. The x - and y -axes are labeled with the atomic numbers. Each cell at coordinate (x, y) represents the correlation between the corresponding elements with atomic numbers x and y . The top row (**a** and **b**) shows the correlations

between embeddings from CrabNet and the bottom row (**c** and **d**) from HotCrab. The left and right columns represent the embeddings extracted from the first and last layer of the models, respectively. Blue represents a high correlation and red represents a high anticorrelation. In **d**, some regions of interest are annotated

the Oliynyk and Magpie CBFVs in Fig. 1 as well. The ability of the CrabNet and HotCrab models to learn such chemical relationships which are comparable to hand-curated CBFVs based solely on the chemical formulae is exciting, and further reaffirms the finding that hand-engineering of features is not needed when training on big data [18].

Moreover, in Fig. 2c we observe a distinct “border” at the element plutonium (with atomic number 94), where the correlation coefficients between the elements suddenly decrease and the patterns become less pronounced.

Additional analysis of the OQMD_Bandgap dataset showed that it does not contain any compounds with elements past plutonium. Due to the fact that the element representations are learned purely by the model from the dataset, their quality depends heavily on the quality of the dataset. Since the model performance depends on the quality of the element representations, by extension, it also then depends on the dataset quality [32].

We define element prevalence as the number of times a certain element has appeared as part of the compounds

in a given dataset. When examining the OQMD_Bandgap dataset, we note that there is an imbalance in element prevalence, with oxygen and copper appearing almost 1.5 times to twice as often, and fluorine, chlorine, bromine and iodine appearing only less than 0.1 times as often as the majority of the other elements in the dataset, respectively. This imbalance in element prevalence is even stronger for other datasets such as the aflow__Egap, castelli, CritExam, mp_e_form and phonons datasets (see supplementary Fig. S-3 in the SI for some example element prevalence plots).

Quantifying Dataset Imbalance

The degree to which a dataset is imbalanced (otherwise referred to as its “evenness”) can be measured using the Shannon equitability index, which is a function of the Shannon entropy of the dataset [62–64]. Shannon entropy is widely used in information theory and can be used to characterize the degree of imbalance in a dataset [65, 66]. The Shannon entropy H is defined in Equation 2 as:

$$H(X) = - \sum_{i=1}^k P(x_i) \log P(x_i) \quad (2)$$

where X is the set of discrete variables $x_i \in \{x_1, \dots, x_n\}$, i is the class, $P(x_i)$ is the proportional abundance of x_i and k is the total number of classes in the dataset.

For a dataset \mathcal{D} of n data occurrences and k distinct chemical elements (classes), each with counts c_i , $P(x_i) = \frac{c_i}{n}$ and the Shannon entropy can thus also be written as Equation 3:

$$H(\mathcal{D}) = - \sum_{i=1}^k \frac{c_i}{n} \log \left(\frac{c_i}{n} \right) \quad (3)$$

For continuity, we note that when $c_i = 0$, it means that no data sample is related to class i in the dataset, and therefore the multiplicand within the summation is defined to be 0. Mathematically, $\lim_{p \rightarrow 0^+} p \log(p) = 0$. The maximum value of $H(\mathcal{D})$ is $\log(k)$. This value occurs when all element classes in the dataset are observed at the same frequency (*i.e.*, the dataset is completely balanced). Therefore, the Shannon entropy $H(\mathcal{D})$ is scaled by $\log(k)$ to finally obtain the Shannon equitability index $E(\mathcal{D})$, which is defined in Equation 4 as:

$$E(\mathcal{D}) = \frac{H(\mathcal{D})}{\log(k)} \quad (4)$$

$E(\mathcal{D})$ ranges between 0 for a maximally imbalanced dataset and 1 for a maximally balanced dataset. The Shannon equitability indices are calculated for the 28 datasets examined in this work and are presented in Table 1. A plot showing the same information can be found in the SI (supplementary Figure S-4). For more information about the datasets, please refer to the CrabNet publication [28].

As can be seen in the table, the datasets studied in this work are not equally balanced in terms of element diversity. The more imbalanced a dataset is in terms of the element prevalence in the chemical compounds, the less likely the models will be able to adequately learn about the elements and their environments. The element embeddings learned for the infrequent elements will therefore be weaker and will not

Table 1 Shannon equitability indices calculated from the training data splits of the 28 reported datasets. Datasets were taken from [28]

Material property dataset	Equitability	Material property dataset	Equitability
castelli	0.823	aflow__ael_bulk_modulus_vrh	0.948
dielectric	0.864	aflow__ael_debye_temperature	0.948
elasticity_log10(G_VRH)	0.953	aflow__ael_shear_modulus_vrh	0.948
elasticity_log10(K_VRH)	0.953	aflow__agl_thermal_conductivity_300K	0.940
expt_gap	0.931	aflow__agl_thermal_expansion_300K	0.944
expt_is_metal	0.930	aflow__Egap	0.920
glass	0.771	aflow__energy_atom	0.917
jdft2d	0.872	CritExam__Ed	0.914
mp_e_form	0.913	CritExam__Ef	0.914
mp_gap	0.916	mp_bulk_modulus	0.923
mp_is_metal	0.916	mp_elastic_anisotropy	0.921
phonons	0.909	mp_e_hull	0.897
steels_yield	0.959	mp_mu_b	0.897
		mp_shear_modulus	0.921
		OQMD_Bandgap	0.976
		OQMD_Energy_per_atom	0.976
		OQMD_Formation_Enthalpy	0.976
		OQMD_Volume_per_atom	0.976

be able to capture as much information about these elements as compared to more frequently occurring elements. This leads to the observed weak correlation patterns between the less frequently seen elements beyond a certain cutoff atomic number in the datasets, as discussed earlier for Fig. 2.

If the weakly learned elements are then encountered during inference time, the model will not be able to make an adequate prediction using the elements' representations. Additionally, if certain elements or element combinations appear more frequently (majority classes) in the datasets as compared to other elements or combinations (minority classes), the model may be biased to better capture the behavior of majority classes at the expense of sacrificing performance on the minority classes. Such a dataset bias may appear in computational or experimental datasets due to the fact that some elements are more commonly studied for certain material applications. On the other hand, certain elements (*e.g.*, rare or unstable elements) naturally occur less frequently and therefore are also contained in fewer compounds and datasets. Certain elements such as noble gases also rarely form compounds with other elements and are therefore rarely reported in materials datasets.

It is therefore important to implement data processing and modeling techniques to address biases as a result of dataset imbalance. Some example techniques include dataset re-sampling, generating synthetic data for imbalanced classes, implementing weighted loss functions that penalize errors for minority classes more, or using alternative loss functions and metrics to evaluate model performance [64, 67, 68]. Additionally, the model architecture can also be tailored to address dataset bias, and certain types of models (such as those based on self-attention or guided attention architectures) have an increased robustness against dataset bias [69, 70].

Lastly, it is worthy to note that while most DL models learn element representations from structured materials datasets, methods such as word2vec and mat2vec use text mining and other natural language processing (NLP) techniques to learn the element embeddings from academic publications [20, 71, 72]. The data present in publications covers a much longer time period and contains a higher diversity in terms of types of compounds, material properties and applications studied. These data are in unstructured form and therefore cannot be used as training data for DL methods such as CrabNet; however, they can easily be used for word2vec and mat2vec. Therefore, these text mining methods are able to learn from a much larger corpus of materials data and are not restricted by the availability of structured datasets. Accordingly, DL models such as CrabNet can benefit by using the pre-trained mat2vec element embeddings and fine-tuning them to new tasks, thereby minimizing the impact of missing elements in the training dataset.

Capturing the Influence of Chemical Environments on Element Representations

In addition to learning the representations of each element, CrabNet and HotCrab can also capture the behavior of the elements when they are present in different chemical environments. Figure 3 shows the two-dimensional projections of the element vectors corresponding to the silicon atom from 2374 different silicon-containing compounds within the OQMD_Bandgap test dataset. The silicon vectors are extracted from the transformed EDM tensors from HotCrab (a onehot-featurized version of CrabNet) and show the transformation of the silicon representations after they are passed through the three successive self-attention layers. For visualization, the vectors are projected down to two dimensions using the uniform manifold approximation and projection (UMAP) method [73]. The resulting points are plotted and colored by three parameters: (1) the fractional abundance of the element silicon in the compound, (2) the predicted property value of the compound (in this case, band gap), and (3) the oxidation state of silicon as predicted by Pymatgen [74]. For more information, please see the Methods.

As can be seen in the plots from the first layer (first row), there is a large number of distinct point clusters, with one major cluster near the center, two medium clusters above and below the center cluster, and many smaller clusters consisting of a few points. The larger clusters are formed because the initial representations of the silicon atoms are very similar to another (due to the learned element embedding of silicon). The similar silicon vectors are thus projected through UMAP into coordinates that lie close together, even though the silicon atoms are present in different chemical environments.

We can observe as well that the clustering in layer one is mostly attributable to the fractional amount, since each cluster consists primarily of points with the same fractional silicon amount. After the second layer, we observe that the points start to become separated into different and recognizable clusters. The clusters are no longer identifiable entirely based on the fractional amount of silicon, and clusters based on the predicted band gap value of the compound and oxidation state of silicon start to emerge. By the end of the third and last layer, we can observe four clusters that are distinguishable by the fractional amount of silicon, the predicted band gap, and the oxidation state of silicon (the clusters are outlined in Fig. 3, bottom left).

More specifically, we observe that the cluster at the bottom-right side of the plot consists mainly of silicon with a fractional amount of around 0.15 to 0.3 (with a few points reaching 0.5), whereas the cluster near the bottom-left contains almost exclusively of silicon with fractional amounts of 0.5 plus a few points above 0.5. The cluster near the top contains regions of silicon with fractional amounts between

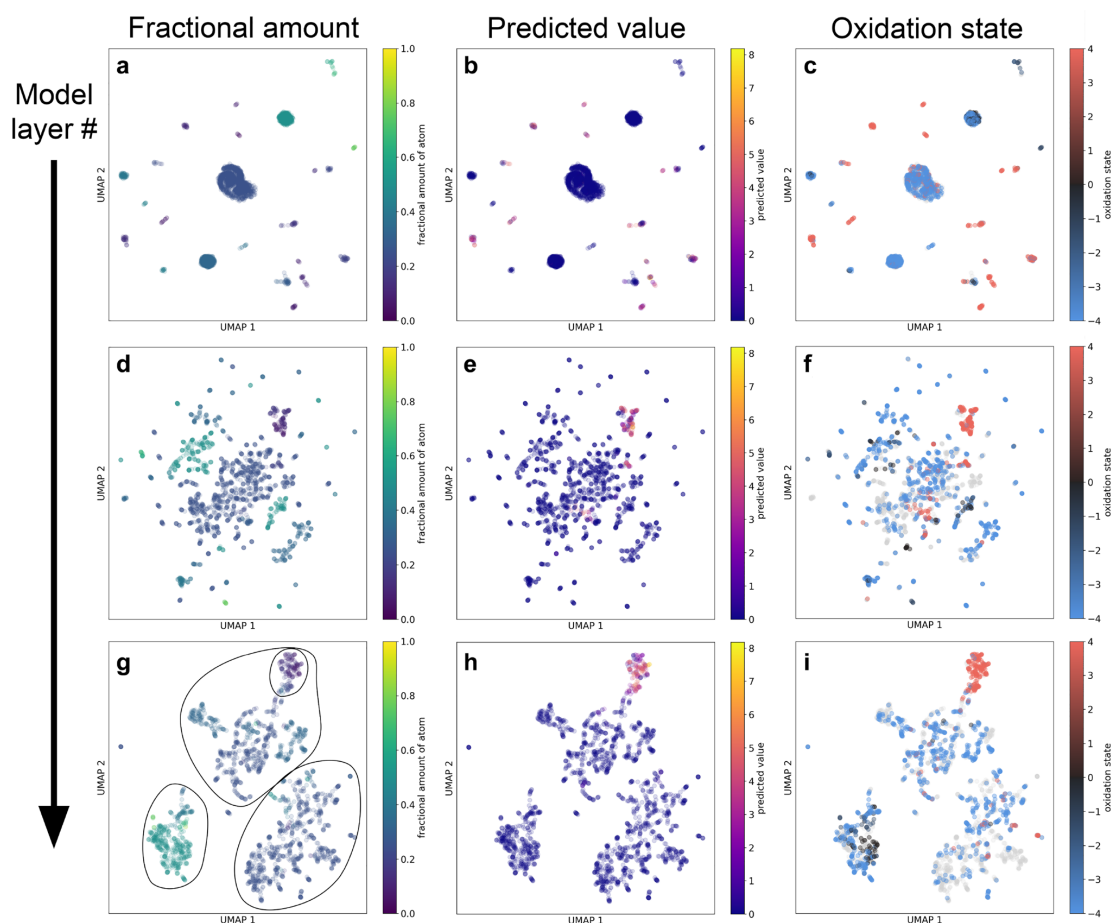


Fig. 3 Vector representations of the silicon element in 2374 different chemical environments and at different layers of the HotCrab model. Each point shows the model-internal representation of the silicon atom, after the information regarding the other atoms in the chemical environment have been introduced via HotCrab through the three attention layers (top row to bottom row). The points are colored by:

(left column) the fractional abundance of silicon, (center column) the predicted value of the compound, and (right column) the predicted oxidation state of silicon, where gray points indicate that the oxidation state was unable to be predicted. Four clusters are outlined in the bottom-left plot

0.3 to 0.4 near the left and right, and around 0.2 to 0.3 in the middle. Near the top of this cluster, a smaller cluster is highlighted which consists mainly of silicon instances with low abundance, between 0.2 and 0. Please note that interactive versions of these plots can be found in the SI together with another example visualization plotted for the element chromium (supplementary Fig. S-5).

In the predicted value plot of the last layer, we observe that only the small cluster near the top contains the silicon element in compounds with a nonzero band gap. Similarly, when examining the oxidation state plot, we note that while most clusters contain a mixture of silicon atoms in several oxidation states, the same cluster near the top consists almost

exclusively of silicon atoms in the +4 oxidation state and very few atoms in other oxidation states. Closer examination reveals that this cluster consists primarily of silicate materials such as Ca_2SiO_4 , CaMgSiO_4 , MgMnSiO_4 , Li_4SiO_4 , $\text{Sr}_3\text{MgSi}_2\text{O}_8$, $\text{Li}_2\text{MgSiO}_4$, and others. Interestingly, while some compounds with silicon in the +4 state are visible in other clusters, these compounds have a zero band gap. This suggests that additional interactions between the elements were captured by HotCrab which lead to these compounds being correctly clustered together with other compounds with zero band gap.

These element behavior plots suggest that for silicon-containing compounds in the OQMD_Bandgap dataset,

the fractional amount and the oxidation state of the silicon atoms are important factors that together determine the band gap of the compounds. By cross-referencing the three plots, we can identify trends between the fractional amount and oxidation state of silicon and relate this information to the predicted band gap of the compounds. On the other hand, the clustering also suggests that there are other interactions between the elements in a compound which are currently not highlighted by the selected properties in Fig. 3. It is our expectation that by examining these interactions, additional insight about the modeling process and element representations can be gained. Moreover, the findings from examining internal representations of elements in this way may suggest additional studies to further improve the understanding of the underlying phenomena governing materials behaviors. Note that while these visualizations were generated using HotCrab, similar results can be obtained using the CrabNet model.

Capturing Globally Unique Representations of Chemical Compounds

In addition to examining the behavior of individual elements in different chemical environments, we can also visualize all of the compounds in a given dataset to uncover additional insights. We extract the internal vector representation of all of the 51242 compounds in the OQMD_Bandgap test dataset from the last self-attention layer of HotCrab, perform dimensionality reduction using UMAP and finally visualize the compounds as shown in Fig. 4. In addition to coloring the plots by the predicted value, prediction error, and number of distinct elements for the compounds, we also highlight the chemical trend between ionic to covalent bonding character within the compounds. This trend is revealed by calculating and visualizing the standard deviation of the Pauling electronegativities of the constituent atoms σ_x in a given compound [75] according to Equation 5:

$$\sigma_x = \sqrt{\frac{\sum_{i=1}^n (\chi_i - \bar{\chi})^2}{n - 1}} \quad (5)$$

where χ_i is the Pauling electronegativity of each element i in the compound (totaling n elements), and $\bar{\chi}$ is the average electronegativity of all elements in the compound. A higher σ_x signifies a more ionic bonding character, and a lower value signifies a more covalent bonding character.

Many clusters with varying sizes are visible in the figure. Some clusters are placed further apart, while some clusters are closer to, or are overlapping other clusters. In particular, the outlined cluster near the right of the figure is of particular interest. This is the only cluster where the compounds with a nonzero band gap are located, as is visible from Fig. 4a. Additionally, it is also within this cluster that

HotCrab makes the largest errors when predicting the band gap value, as seen in Fig. 4b. For the other compounds, the prediction errors of HotCrab are close to zero. Even through a small proportion of model predictions have larger errors, the overall model performance is very good and is comparable with, or better than, other state-of-the-art models [28]. This superior performance of CrabNet and HotCrab models when predicting properties with a defined cutoff (such as the cutoff of 0 eV in this case for band gap) is likely attributed to the prediction of element-logits in the modeling process. These element-logits are used to weight the final model predictions in CrabNet and HotCrab to improve the model accuracy [28].

Notably, we also observe from Fig. 4a, c and d that the band gap only partially depends on the bond nature of the compound and on the number of unique elements in the compound. While most of the compounds in the cluster of interest exhibit more ionic bond characters, there are also other clusters with similar bond character that do not have a nonzero band gap. Similarly, it appears that the compounds with a nonzero band gap mainly contain four or five unique elements; however, there are also other compounds with these numbers of unique elements which have a zero band gap.

Here, we do note that while UMAP can reveal structures and patterns within high-dimensional data, it generally emphasizes local structure at the expense of global structure. Therefore, for the UMAP visualizations shown in this work, it is more appropriate to interpret the local structure (*e.g.*, the elements or compounds present within individual clusters in Fig. 3 and 4) than the global structure. While the number of local neighbors considered can be specified as a hyperparameter in UMAP, a trade-off is made between preserving local versus global structure. Therefore, the distances between elements and compounds within a single cluster are more meaningful than inter-cluster distances in the UMAP visualizations. Lastly, we note that while these visualizations were generated based on the test dataset using HotCrab, similar results can be obtained using CrabNet or the training dataset.

Visualizing the Training Progress

Beyond visualizing the element and compound representations from CrabNet after training, it is also possible to access the self-attention matrices of the CrabNet encoding layers to observe the model learning process during training. The attention matrices (commonly referred to as the attention maps) contain information regarding how each element (rows) is influenced by all other elements in the compound as well as itself (columns). The values in the attention maps are the attention scores and are used in the encoder to update the element representations. An attention score of

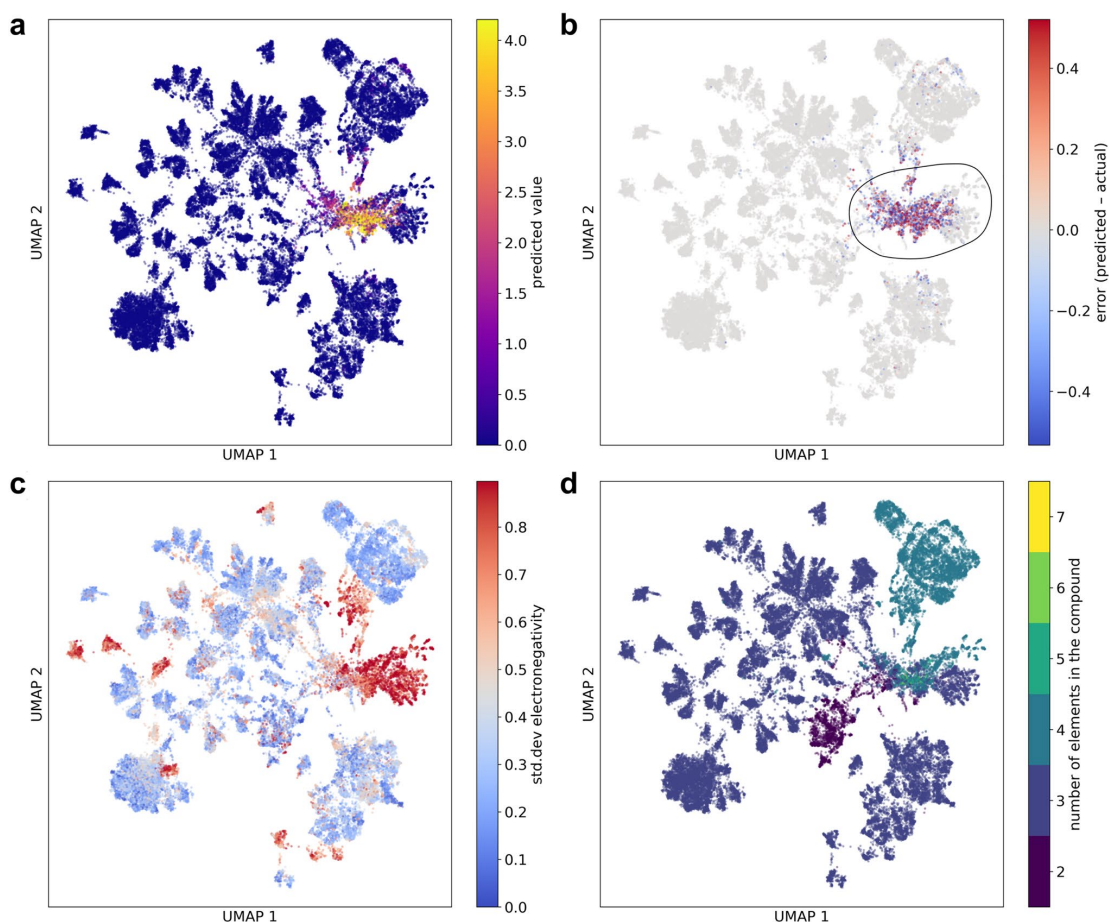


Fig. 4 Global representations of the 51242 compounds in the OQMD_Bandgap test dataset, extracted from layer three of HotCrab, embedded down to two dimensions using UMAP and colored by the parameters: **a** the predicted value of the compound (band gap); **b** the prediction error ($\hat{y} - y$); **c** the bond character of the compounds rang-

ing from more covalent (blue) to more ionic (red) as measured by the standard deviations in the Pauling electronegativities of the constituent elements; and **d** the number of distinct elements in the compound. A cluster of interest is outlined in the plot at the top-right

zero means that the element in the column is completely ignored when updating the element's representation in that row. Conversely, a score of one means that the entire update is based solely on that column's element.

In the CrabNet publication [28], example attention maps were shown for compounds after the model has finished training. Here, we extend this approach by visualizing the CrabNet attention maps during the model training process in the form of attention video clips (see SI files for examples). This is achieved by saving the attention matrices from the model encoder layers after every mini-step in the training process and generating a video to show the learning progress. Fig. 5 shows a snapshot of two example attention

videos obtained at the end of model training. The attention maps from the first encoding layer of CrabNet are plotted as heatmaps in the left column, while the right column shows the predicted values from the model against the target value at every mini-step. This process is performed at every mini-step in the training process, and the resulting plots are merged into a video clip which shows the learning progress of the model throughout training.

From the attention maps, we can observe that some elements are considered less relevant in the determination of the material property, whereas some elements are considered very relevant. Also we can note that individual attention heads pay attention to different element-element interactions

5 INTERPRETABLE DEEP LEARNING WITH CRABNET

Integrating Materials and Manufacturing Innovation

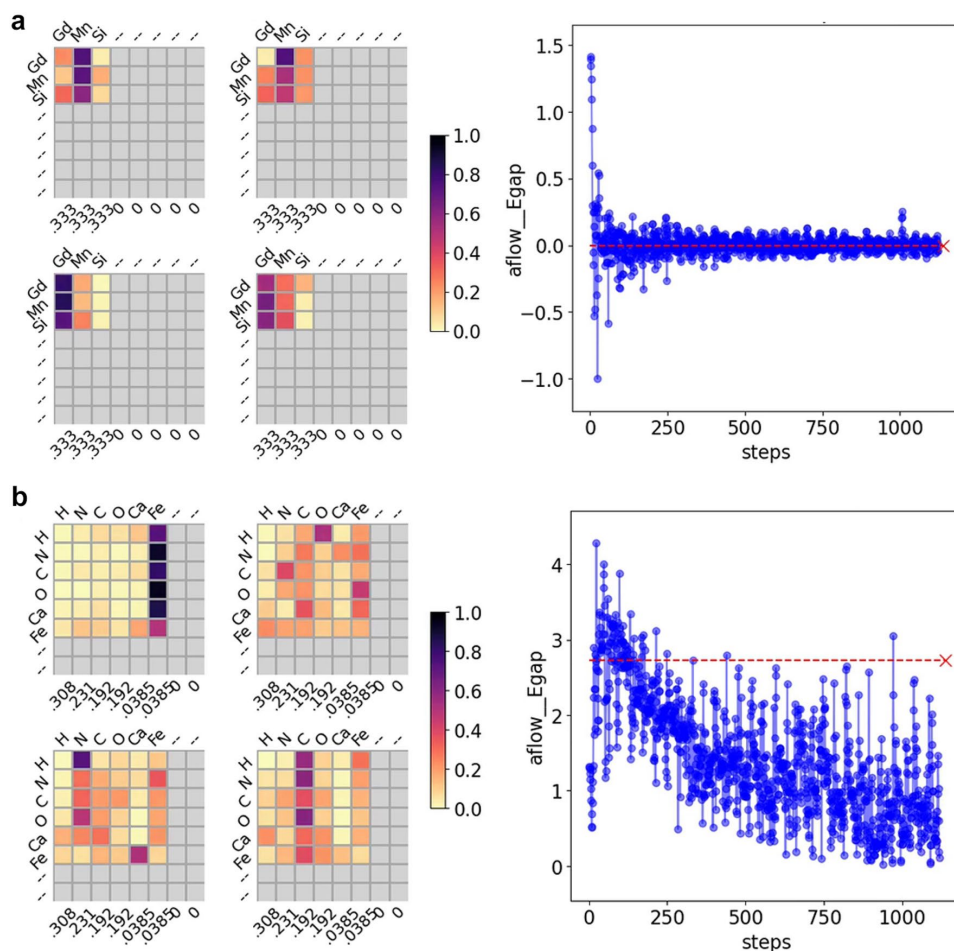


Fig. 5 Snapshots of attention videos for observing the training progress of CrabNet using two example compounds **a** $\text{Gd}_1\text{Mn}_1\text{Si}_1$ and **b** $\text{C}_5\text{Ca}_1\text{Fe}_1\text{H}_8\text{N}_6\text{O}_5$ from the validation data split of the `aflow_Egap` dataset. The left plots show the attention maps of the four attention heads at the first attention layer, where the x axis of each heatmap is

labeled with the fractional amount of the elements and the other axes are labeled with the element symbol. The right plots show the model predictions (blue) for the compounds, evaluated after each training mini-step throughout the whole training process. The true property value (target) is represented with the red "X" and the dotted line

in the compound, as is visible by the significantly different attention patterns in the plots. Throughout the training process, the attention pattern for each head remains relatively fixed after a few mini-steps, indicating that the model discovers a pattern for recognizing inter-element interactions early on in the training process, which it then continues to refine as more training steps are taken.

For the top compound, we can observe that while the model initially over- and underestimates the property value early on in the training, it learns to correct the error and finally achieves a low prediction error towards the end of

training. Conversely, for the bottom compound, we observe that while the model initially correctly estimates the property value of the compound, the predicted value decreases and the estimation error increases throughout training, with the error finally plateauing towards the end of the training. By examining the attention heatmaps for this compound, we notice that attention head 1 shows a significantly different behavior as compared to the other attention heads. It dedicates almost all of its attention to the element iron, while the other attention heads capture many more inter-element interactions. It may be interesting to investigate further to

find out if CrabNet is misrepresenting the interactions from the iron element with the other elements and thus making the prediction error, or if another phenomenon is contributing to the prediction error on this compound.

By observing the element groups and inter-elemental interactions that CrabNet pays attention to for each material property throughout the training process, we may be able to gain additional insight about which relevant elements and interactions contribute significantly to the material property. Similarly, in the case where the model does not make a good property prediction or fails to learn a specific material property, these attention videos can be informative in showing when, where, and how the model fails. Additionally, since the element representations in a compound are updated according to the attention scores, it would be interesting to train CrabNet on material properties where the property has a high sensitivity to changes in elemental prevalence. An example of this is in the case of dopants, where a small change in the dopant amount can significantly influence a material's electrical [15, 76, 77], mechanical [17, 78–80], and thermal properties [81–84]. Finally, it may be interesting to expand the studied materials to include co-doped materials and use the attention videos to visualize the complex inter-elemental interactions between the co-dopants and the host elements.

Conclusion

In this work, we examined the CrabNet model through the use of several built-in model interpretability methods in order to visualize the data featurization and modeling process. We demonstrated that CrabNet can adequately capture the chemical behavior of compounds in a dataset by using the vector representations of their constituent elements. The element representations can be learned entirely from the training data on a per-property basis, and contain rich information about the elements and their chemical trends. Additionally, we examined dataset imbalance, its relation to the quality of learned representations, and the limitations that imbalanced datasets may ultimately impose on the modeling processes.

The element and compound vectors can be projected using UMAP into distinguishable clusters which can then be visualized and characterized by the element stoichiometry, local chemical environment and oxidation state of the elements, or by the bond behavior of the compounds. Lastly, the examination of the self-attention matrices during model training in the form of attention videos can be used to further understand the modeling process, debug potential model or dataset errors, or gain additional insights about chemical interactions within a given compound.

The model interpretability techniques presented in this work will enable materials science practitioners to not only visualize a specific element's behavior within different chemical environments, but also to obtain a global view of the chemical compounds, behaviors and trends within a larger dataset. The ability of CrabNet to adequately model and express the complex chemical behaviors and interactions of elements and compounds based solely on learning from data is encouraging. With the addition of model interpretability methods to CrabNet, the findings and intuitions presented in this work may lead to further insightful and interesting research. Specifically, we believe that follow-up works may fall into one of these three general directions:

1. **Learning and representing elements and compounds.** Our work has shown that it is possible to visualize CrabNet's internal representations of elements and compounds via techniques such as UMAP. However, it would be interesting to further investigate why CrabNet's representations of some of these elements or compounds lead to them being placed into the same cluster or not, despite the fact that these elements and compounds are similar to each other in terms of identity and/or chemical environment. This may also be combined with a more detailed examination of the attention videos and how the attention mechanism in CrabNet leads to the updating of the element representations for each compound.
2. **Examination of individual attention head behaviors.** This work used the EDM (element-derived matrix) data from CrabNet to examine the element and compound representations within CrabNet. CrabNet utilizes four self-attention heads to model element-element interactions, the results of which are then concatenated and transformed back to an updated EDM matrix. As such, the EDM is a pooled representation of the compounds. It would be interesting to further examine the per-head modeling of the compounds, as it has been shown that each head can capture different types of inter-element interactions and thus may give additional insight to the modeling process within CrabNet.
3. **Discovery of additional inter-element interactions.** From the analyses presented in this study, it is clear that while some changes in the material property (*e.g.*, band gap) can be explained by certain properties of the compounds (such as element stoichiometry, number of unique elements, and/or bond character), there are additional behaviors that govern the material property. These additional interactions are also adequately modeled by CrabNet, since it can predict a wide range of material properties with low errors. Examining the modeling process of these behaviors within CrabNet may lead to

an improved understanding of the complex phenomena underlying material properties.

Further research to answer these and subsequent questions may allow us to gain additional insights about the behaviors and properties of elements and materials, improve our understanding of models such as CrabNet, increase our confidence in the use of data-driven methods, and ultimately, accelerate the adoption of deep learning and machine learning in materials science.

Methods

Adaptation of CrabNet Model

The CrabNet model and material property datasets as originally reported were used as the basis for this study [28]. Fully trained model weights for both CrabNet and HotCrab were obtained from [57]. In order to obtain the EDMs containing the elements and compounds data used in this study, custom function hooks were implemented in PyTorch. These hooks were attached to the CrabNet model architecture to allow access to the model-internal data during training and inference.

The source code as well as the data that were used and generated in this study can be found on the updated CrabNet GitHub repository [58]. In addition, we provide detailed instructions for the use and reproduction of our reported results. Please note that due to the prohibitively large size of the stored attention matrices used in the attention videos, it is not possible to provide these for download. However, instructions and scripts are provided for generating these matrices and videos.

All experiments, unless otherwise noted, were performed on a workstation equipped with an Intel i7-8700K CPU, 32 GB of DDR4 RAM, and one Nvidia RTX 2080 GPU.

Element Embeddings

Element embeddings for pure elements were generated on a per-property basis. To do this, an EDM consisting of all of the elements from hydrogen to oganesson was generated (with each row representing one element). Then, for each material property, the corresponding CrabNet or HotCrab model was loaded and the model hooks attached. The EDM was then passed through the network and the modified EDM at the output of the element embedding layer was obtained and detached from the model graph. This resulting EDM contains the property-specific element embeddings of all of

the elements. Thus, each element was represented by a vector with the shape $(1, d_{\text{model}})$, where d_{model} is the size of the embedding. Element embeddings for Oliynyk, Magpie, and mat2vec were obtained from the original publications [18].

Compound Embeddings

Compound embeddings were obtained in a similar fashion to element embeddings. Instead of generating an EDM from pure elements, the EDMs were generated from the actual chemical formulae from the datasets and collated in batches using the model data loader. Model hooks were then attached to the CrabNet and HotCrab models and enabled during model inference. The transformed EDMs after each of the three self-attention layers of the CrabNet models were then collected.

The obtained compound EDMs have the shape of $(n_{\text{compounds}}, n_{\text{elements}}, d_{\text{model}})$, where $n_{\text{compounds}}$ is the total number of compounds in the dataset, n_{elements} is the maximum number of elements per compound, and d_{model} is the size of the embedding. Thus, each compound in the EDM is represented by one tensor slice with the dimensions $(1, n_{\text{elements}}, d_{\text{model}})$. Due to the fact that different compounds within the same dataset may contain a different number of elements, the extra rows of the EDMs were zero-filled to indicate no elements present. In order to ensure that the compound embeddings are comparable with each other using UMAP, the three-dimensional compound EDMs were collapsed to two dimensions $(n_{\text{compounds}}, 1, d_{\text{model}})$ by calculating summary statistics (such as sum, range, variance) of the EDM columns across the elements dimension.

Dimensionality Reduction

CrabNet uses vectors with a d_{model} dimension of 512 to represent chemical elements and compounds in the input data. It would be infeasible to try to visualize all 512 dimensions. Therefore, dimensionality reduction was applied to the vector representations to transform the vectors into two-dimensional space for visualization.

Three common methods for dimensionality reduction were tested: principal component analysis (PCA), t-distributed stochastic neighbor embedding (t-SNE), and uniform manifold approximation and projection (UMAP) [73, 85, 86]. Compared to t-SNE and PCA, UMAP revealed more visually distinct clusters for the data presented in this work. Therefore, UMAP was chosen as the dimensionality reduction method. The random seed was fixed so that each initialization of the UMAP method produces the same results. For element embeddings, the rows of the EDMs with dimensions $(1, d_{\text{model}})$ are transformed using UMAP. For the compound embeddings, the matrices

corresponding to each compound were first collapsed as described above, and the resulting representations with dimensions $(1, d_{\text{model}})$ for each compound were transformed using UMAP.

Oxidation State Estimation

Oxidation states for elements in the compounds were estimated using the Pymatgen package (version 2022.0.8) using the chemical formulae of the compounds. The built-in functions for assigning oxidation states were used, which are based on charge-balancing heuristics and use the most probable oxidation states as determined based on the compounds in the Inorganic Crystal Structure Database [74].

Attention Video Generation

Custom function hooks were programmed and attached to a newly-initialized CrabNet model. During training of CrabNet, the attention matrices of every CrabNet encoder layer was extracted from the model and saved into a compressed Zarr array on disk. The model predictions for the properties were also generated and saved. This procedure is performed after every mini-step during the training process (corresponding to each mini-batch of data). The plots were then generated for each mini-step and merged together using the software FFMPEG to create the attention videos. Due to the large amount of storage and computing power required to store and process the attention matrices, these tasks were performed on a high-performance computing cluster.

Supplementary Information The online version contains supplementary material available at <https://doi.org/10.1007/s40192-021-00247-y>.

Acknowledgements The authors thank the Berlin International Graduate School in Model and Simulation based Research as well as the German Academic Exchange Service RISE program for their financial support. Special thanks is given to Dr. Steven K. Kauwe, Pay Gießelmann and Joris Weigert for the insightful discussions. Computing resources were graciously provided by the HPC-Cluster at the Institut für Mathematik, Technische Universität Berlin, by the HPC Resource in the Core Facility for Advanced Research Computing at Case Western Reserve University as well as by the Google TPU Research Cloud (TRC) program. In addition, the authors express their gratitude to the open-source software community for developing the excellent tools used in this research, including but not limited to Python, Pandas, NumPy, matplotlib, scikit-learn, PyTorch, Zarr, and FFMPEG.

Funding Open Access funding enabled and organized by Projekt DEAL.

Compliance with Ethical Standards

Conflicts of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Ramprasad R, Batra R, Pilia G, Mannodi-Kanakkithodi A, Kim C (2017) Machine learning in materials informatics: recent applications and prospects. *npj Comput Mater* 3(1):60
- Schmidt J, Marques MRG, Botti S, Marques MAL (2019) Recent advances and applications of machine learning in solid-state materials science. *npj Comput Mater* 5(1):83
- Gomes CP, Selman B, Gregoire JM (2019) Artificial intelligence for materials discovery. *MRS Bull* 44(7):538–544
- Isayev O, Tropsha A, Curtarolo S (eds) (2019) *Materials informatics: methods, tools, and applications*. Wiley, USA
- DeCost BL, Hattrick-Simpers JR, Trautt Z, Kusne AG, Campo E, Green ML (2020) Scientific AI in materials science: a path to a sustainable and scalable paradigm. *Mach Learn: Sci Technol* 1(3):033001
- Stein HS, Gregoire JM (2019) Progress and prospects for accelerating materials science with automated and autonomous workflows. *Chem Sci* 10(42):9640–9649
- Morgan D, Jacobs R (2020) Opportunities and challenges for machine learning in materials science. *Annu Rev Mater Res* 50(1):71–103
- Zitnick CL, Chanussot L, Das A, Goyal S, Heras-Domingo J, Ho C, Hu W, Lavril T, Palizhati A, Riviere M, Shuaibi M, Sriram A, Tran K, Wood B, Yoon J, Parikh D, Ulissi Z (2020) An introduction to electrocatalyst design using machine learning for renewable energy storage. <http://arxiv.org/abs/2010.09435v1>
- Sparks TD, Kauwe SK, Parry ME, Tehrani AM, Brgoch J (2020) Machine learning for structural materials. *Ann Rev Mater Res* 50:27
- Pilia G (2021) Machine learning in materials science: from explainable predictions to autonomous design. *Comput Mater Sci* 193:110360
- Oliynyk AO, Antono E, Sparks TD, Ghadbeigi L, Gaultois MW, Meredig B, Mar A (2016) High-throughput machine-learning-driven synthesis of full-heusler compounds. *Chem Mater* 28(20):7324–7331
- Ward L, Agrawal A, Choudhary A, Wolverton C (2016) A general-purpose machine learning framework for predicting properties of inorganic materials. *npj Comput Mater* 2(1):16028
- Pilia G, Mannodi-Kanakkithodi A, Uberuaga BP, Ramprasad R, Gubernatis JE, Lookman T (2016) Machine learning band-gaps of double perovskites. *Sci Rep* 6:19375
- Dunn A, Wang Q, Ganose A, Dopp D, Jain A (2020) Benchmarking materials property prediction methods: the Matbench test set and Automatminer reference algorithm. *npj Comput Mater* 6(1):138

5 INTERPRETABLE DEEP LEARNING WITH CRABNET

Integrating Materials and Manufacturing Innovation

15. Kauwe SK, Graser J, Murdock RJ, Sparks TD (2020) Can machine learning find extraordinary materials? *Comput Mater Sci* 174:109498
16. Graser J, Kauwe SK, Sparks TD (2018) Machine learning and energy minimization approaches for crystal structure predictions: a review and new horizons. *Chem Mater* 30(11):3601–3612
17. Tehrani AM, Oliynyk AO, Parry M, Rizvi Z, Couper S, Lin F, Miyagi L, Sparks TD, Brgoch J (2018) Machine learning directed search for ultraincompressible, superhard materials. *J Am Chem Soc* 140(31):9844–9853
18. Murdock RJ, Kauwe SK, Wang AY-T, Sparks TD (2020) Is domain knowledge necessary for machine learning materials properties? *Integr Mater Manuf Innov* 9(3):221–227
19. Choudhary K, DeCost B, Tavazza F (2018) Machine learning with force-field-inspired descriptors for materials: fast screening and mapping energy landscape. *Phys Rev Mater* 2(8):083801
20. Tshitoyan V, Dagdelen J, Weston L, Dunn A, Rong Z, Kononova O, Persson KA, Ceder G, Jain A (2019) Unsupervised word embeddings capture latent knowledge from materials science literature. *Nature* 571(7763):95–98
21. Kauwe SK, Graser J, Vazquez A, Sparks TD (2018) Machine learning prediction of heat capacity for solid inorganics. *Integr Mater Manuf Innov* 7(2):43–51
22. Kauwe SK, Welker T, Sparks TD (2020) Extracting knowledge from DFT: experimental band gap predictions through ensemble learning. *Integr Mater Manuf Innov* 9(3):213–220
23. Jha D, Ward L, Paul A, Liao W-K, Choudhary A, Wolverton C, Agrawal A (2018) ElemNet: deep learning the chemistry of materials from only elemental composition. *Sci Rep* 8(1):17593
24. Xie T, Grossman JC (2018) Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Phys Rev Lett* 120(14):145301
25. Chen C, Ye W, Zuo Y, Zheng C, Ong SP (2019) Graph networks as a universal machine learning framework for molecules and crystals. *Chem Mater* 31(9):3564–3572
26. Klicpera J, Giri S, Margraf JT, Günnemann S (2020) Fast and uncertainty-aware directional message passing for non-equilibrium molecules. <https://arxiv.org/abs/2011.14115>
27. DeCost B, Choudhary K (2021) Atomistic line graph neural network for improved materials property predictions. *npj Comput Mater* 7(1):185
28. Wang AY-T, Kauwe SK, Murdock RJ, Sparks TD (2021) Compositionally restricted attention-based network for materials property predictions. *npj Comput Mater* 7(1):77
29. Goodall REA, Lee AA (2020) Predicting materials properties without crystal structure: deep representation learning from stoichiometry. *Nat Commun* 11(1):6280
30. Kong S, Guevarra D, Gomes CP, Gregoire JM (2021) Materials representation and transfer learning for multi-property prediction. *Appl Phys Rev* 8(2):021409
31. Clement CL, Kauwe SK, Sparks TD (2020) Benchmark AFLOW data sets for machine learning. *Integr Mater Manuf Innov* 9(2):153–156
32. Wang AY-T, Murdock RJ, Kauwe SK, Oliynyk AO, Gurlo A, Brgoch J, Persson KA, Sparks TD (2020) Machine learning for materials scientists: an introductory guide toward best practices. *Chem Mater* 32(12):4954–4965
33. Henderson AN, Kauwe SK, Sparks TD (2021) Benchmark datasets incorporating diverse tasks, sample sizes, material systems, and data heterogeneity for materials informatics. *Data Brief* 37:107262
34. Meredig B (2017) Industrial materials informatics: analyzing large-scale data to solve applied problems in R&D, manufacturing, and supply chain. *Curr Opin Solid State Mater Sci* 21(3):159–166
35. Lipton ZC (2018) The mythos of model interpretability. *Queue* 16(3):31–57
36. Himanen L, Geurts A, Foster AS, Rinke P (2019) Data-driven materials science: status, challenges, and perspectives. *Adv Sci* 6(21):1900808
37. Rudin C (2019) Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat Mach Intell* 1(5):206–215
38. Kolyshkina I, Simoff S (2019) Interpretability of machine learning solutions in industrial decision engineering. In: Data mining (T. D. Le, K.-L. Ong, Y. Zhao, W. H. Jin, S. Wong, L. Liu, and G. Williams, eds.), vol. 1127 of communications in computer and information science, pp. 156–170, Singapore: Springer Singapore
39. Linardatos P, Papastefanopoulos V, Kotsiantis S (2021) Explainable AI: a review of machine learning interpretability methods. *Entropy (Basel, Switzerland)* 23(1):18
40. Gilpin LH, Bau D, Yuan BZ, Bajwa A, Specter M, Kagal L (2018) Explaining explanations: an overview of interpretability of machine learning. In: 2018 IEEE 5th international conference on data science and advanced analytics (DSAA), pp 80–89, IEEE
41. Smilkov D, Thorat N, Nicholson C, Reif E, Viégas FB, Wattenberg M (2016) Embedding projector: interactive visualization and interpretation of embeddings. <http://arxiv.org/abs/1611.05469v1>
42. Liu S, Bremer P-T, Thiagarajan JJ, Srikumar V, Wang B, Livnat Y, Pascucci V (2018) Visual exploration of semantic relationships in neural word embeddings. *IEEE Trans Visual Comput Graphics* 24(1):553–562
43. van Aken B, Winter B, Löser A, Gers FA (2020) VisBERT: Hidden-state visualizations for transformers. In: Companion proceedings of the web conference 2020 (A. E. F. Seghrouchni, G. Sukthankar, T.-Y. Liu, and M. van Steen, eds.), (New York, NY, USA), pp 207–211, ACM
44. Vinyals O, Babuschkin I, Czarnecki WM, Mathieu M, Dudzik A, Chung J, Choi DH, Powell R, Ewalds T, Georgiev P, Oh J, Horgan D, Kroiss M, Danihelka I, Huang A, Sifre L, Cai T, Agapiou JP, Jaderberg M, Vezhnevets AS, Leblond R, Pohlen T, Dalibard V, Budden D, Sulsky Y, Molloy J, Paine TL, Gulcehre C, Wang Z, Pfaff T, Wu Y, Ring R, Yogatama D, Wünsch D, McKinney K, Smith O, Schaul T, Lillicrap T, Kavukcuoglu K, Hassabis D, Apps C, Silver D (2019) Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575(7782):350–354
45. Puittua E, Veith EMSP (2020) Explainable Reinforcement Learning: A Survey. In: Machine learning and knowledge extraction (A. Holzinger, P. Kieseberg, A. M. Tjoa, and E. Weippl, eds.), vol. 12279 of Lecture Notes in Computer Science, pp 77–95, Cham: Springer International Publishing
46. Heuillet A, Couthouis F, Díaz-Rodríguez N (2021) Explainability in deep reinforcement learning. *Knowl-Based Syst* 214:106685
47. Lapuschkin S (2018) Opening the machine learning black box with Layer-wise Relevance Propagation. PhD thesis, Technische Universität Berlin, Berlin, Germany
48. Chefer H, Gur S, Wolf L (2020) Transformer interpretability beyond attention visualization. <http://arxiv.org/abs/2012.09838v2>
49. Chen J, Lu Y, Yu Q, Luo X, Adeli E, Wang Y, Lu L, Yuille AL, Zhou Y (2021) TransUNet: transformers make strong encoders for medical image segmentation. <http://arxiv.org/abs/2102.04306v1>
50. Khan S, Naseer M, Hayat M, Zamir SW, Khan FS, Shah M (2021) Transformers in vision: a survey. <http://arxiv.org/pdf/2101.01169v3>
51. Kailkhura B, Gallagher B, Kim S, Hiszpanski A, Han TY-J (2019) Reliable and explainable machine-learning methods for accelerated material discovery. *npj Comput Mater* 5(1):221
52. Roscher R, Bohn B, Duarte MF, Garcke J (2020) Explainable machine learning for scientific insights and discoveries. *IEEE Access* 8:42200–42216

53. Ribeiro MT, Singh S, Guestrin C (2016) Why should I trust you?: explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining – KDD '16 (B. Krishnapuram, M. Shah, A. Smola, C. Aggarwal, D. Shen, and R. Rastogi, eds.), (New York, NY, USA), pp. 1135–1144, ACM Press
54. Lundberg S, Lee S-I (2017) A unified approach to interpreting model predictions. <http://arxiv.org/abs/1705.07874v2>
55. Shapley LS (1953) A Value for n-person games. In: contributions to the theory of games (AM-28), Volume II (H. W. Kuhn and A. W. Tucker, eds.), Annals of Mathematics Studies, pp. 307–318, Princeton, NJ: Princeton University Press
56. Hastie T, Tibshirani R, Friedman JH (2009) The elements of statistical learning: Data mining, inference, and prediction. Springer Series in Statistics. Springer, New York
57. Wang AY-T, Kauwe SK, Murdock RJ, Sparks TD (2021) Trained network weights for the paper, Compositionally restricted attention-based network for materials property predictions (CrabNet). <https://doi.org/10.5281/zenodo.4633866>
58. Wang AY-T, Kauwe SK (2020) Online GitHub repository for the paper, compositionally-restricted attention-based network for materials property prediction. <https://github.com/anthony-wang/CrabNet>
59. Saal JE, Kirklin S, Aykol M, Meredig B, Wolverton C (2013) Materials design and discovery with high-throughput density functional theory: the open quantum materials database (OQMD). *JOM* 65(11):1501–1509
60. Kirklin S, Saal JE, Meredig B, Thompson A, Doak JW, Aykol M, Rühl S, Wolverton C (2015) The open quantum materials database (OQMD): assessing the accuracy of DFT formation energies. *npj Computational Materials* 1(1):15010
61. Hegde VI, Borg CKH, Rosario Z, Kim Y, Hutchinson M, Antono E, Ling J, Saxe P, Saal JE, Meredig B (2020) Reproducibility in high-throughput density functional theory: a comparison of AFLOW, Materials Project, and OQMD. <http://arxiv.org/pdf/2007.01988v1>
62. Bonachela JA, Hinrichsen H, Muñoz MA (2008) Entropy estimates of small data sets. *J Phys A: Math Theor* 41(20):202001
63. Hong C, Ghosh R, Srinivasan S (2016) Dealing with class imbalance using thresholding. <http://arxiv.org/pdf/1607.02705v1>
64. Tahir MAUH, Asghar S, Manzoor A, Noor MA (2019) A classification model for class imbalance dataset using genetic programming. *IEEE Access* 7:71013–71037
65. Shannon CE (1948) A mathematical theory of communication. *Bell Syst Tech J* 27(3):379–423
66. Shannon CE (1948) A mathematical theory of communication. *Bell Syst Tech J* 27(4):623–656
67. Li Y, Vasconcelos N (2019) REPAIR: removing representation bias by dataset resampling. In: 2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR) (CVPR Editors, ed.), pp 9564–9573, IEEE
68. Esposito C, Landrum GA, Schneider N, Stiefl N, Riniker S (2021) GHOST: adjusting the decision threshold to handle imbalanced data in machine learning. *J Chem Inf Model* 61(6):2623–2640
69. Li K, Wu Z, Peng K-C, Ernst J, Fu Y (2018) Tell me where to look: guided attention inference network,” in 2018 IEEE/CVF conference on computer vision and pattern recognition, pp 9215–9223, IEEE
70. Rodriguez AC, D'Aronco S, Schindler K, Wegner JD (2020) Privileged pooling: better sample efficiency through supervised attention. <http://arxiv.org/abs/2003.09168v3>
71. Kim E, Huang K, Tomala A, Matthews S, Strubell E, Saunders A, McCallum A, Olivetti E (2017) Machine-learned and codified synthesis parameters of oxide materials. *Sci Data* 4:170127
72. Weston L, Tshitoyan V, Dagdelen J, Kononova O, Trewartha A, Persson KA, Ceder G, Jain A (2019) Named entity recognition and normalization applied to large-scale information extraction from the materials science literature. *J Chem Inform Model* 59:3692
73. McInnes L, Healy J, Saul N, Großberger L (2018) UMAP: uniform manifold approximation and projection. *J Open Sour Softw* 3(29):861
74. Ong SP, Richards WD, Jain A, Hautier G, Kocher M, Cholia S, Gunter D, Chevrier VL, Persson KA, Ceder G (2013) Python materials genomics (pymatgen): a robust, open-source python library for materials analysis. *Comput Mater Sci* 68:314–319
75. Hargreaves CJ, Dyer MS, Gaultois MW, Kurlin VA, Rosseinsky MJ (2020) The earth mover's distance as a metric for the space of inorganic compositions. *Chem Mater* 32(24):10610–10620
76. Glaudell AM, Cochran JE, Patel SN, Chabinyc ML (2015) Impact of the doping method on conductivity and thermopower in semiconducting polythiophenes. *Adv Energy Mater* 5(4):1401072
77. Zhang SB (2002) The microscopic origin of the doping limits in semiconductors and wide-gap materials and recent developments in overcoming these limits: a review. *J Phys: Condens Matter* 14(34):R881–R903
78. Sheng L, Wang L, Xi T, Zheng Y, Ye H (2011) Microstructure, precipitates and compressive properties of various holmium doped NiAl/Cr(Mo, Hf) eutectic alloys. *Mater Design* 32(10):4810–4817
79. Tehrani AM, Oliynyk AO, Rizvi Z, Lotfi S, Parry M, Sparks TD, Brgoch J (2019) Atomic substitution to balance hardness, ductility, and sustainability in molybdenum tungsten borocarbide. *Chem Mater* 31(18):7696–7703
80. Mihailovich and Parpia (1992) Low temperature mechanical properties of boron-doped silicon. *Phys Rev Lett* 68(20):3052–3055
81. Qu Z, Sparks TD, Pan W, Clarke DR (2011) Thermal conductivity of the gadolinium calcium silicate apatites: effect of different point defect types. *Acta Mater* 59(10):3841–3850
82. Sparks TD, Fuierer PA, Clarke DR (2010) Anisotropic thermal diffusivity and conductivity of La-doped strontium niobate Sr2Nb2O7. *J Am Ceram Soc* 93(4):1136–1141
83. Grimvall G (1999) Thermophysical Properties of Materials. Amsterdam: North Holland, 1 ed
84. Gaumé R, Viana B, Vivien D, Roger J-P, Fournier D (2003) A simple model for the prediction of thermal conductivity in pure and doped insulating crystals. *Appl Phys Lett* 83(7):1355–1357
85. Pearson K (1901) On lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin Phil Magazine *J Sci* 2(11):559–572
86. van der Maaten L, Hinton G (2008) Visualizing data using t-SNE. *J Mach Learn Res* 9:2579–2605

Supplementary Information for the article “CrabNet for Explainable Deep Learning in Materials Science: Bridging the Gap Between Academia and Industry”

Authors: Anthony Yu-Tung Wang¹, Mahamad Salah Mahmoud², Mathias Czasny¹, and Aleksander Gurlo¹

Affiliations:

- (1) Technische Universität Berlin, Institute of Material Science and Technology, Fachgebiet Keramische Werkstoffe / Chair of Advanced Ceramic Materials, Straße des 17. Juni 135, 10623 Berlin, Germany
- (2) Department of Computer and Data Sciences, Case Western Reserve University, Cleveland, Ohio 44106, USA

Correspondence: Correspondence should be sent to Anthony Yu-Tung Wang at anthony.wang@ceramics.tu-berlin.de

DOI: [10.1007/s40192-021-00247-y](https://doi.org/10.1007/s40192-021-00247-y)

URL of CrabNet GitHub with source code: <https://github.com/anthony-wang/CrabNet>

DOI of trained CrabNet model weights: <https://doi.org/10.5281/zenodo.4633866>

Note: all the ZIP files mentioned in this document (**ESM1.zip** to **ESM5.zip**) are available for download from figshare at the following address:
<https://doi.org/10.6084/m9.figshare.16837999>

Supplementary Figures

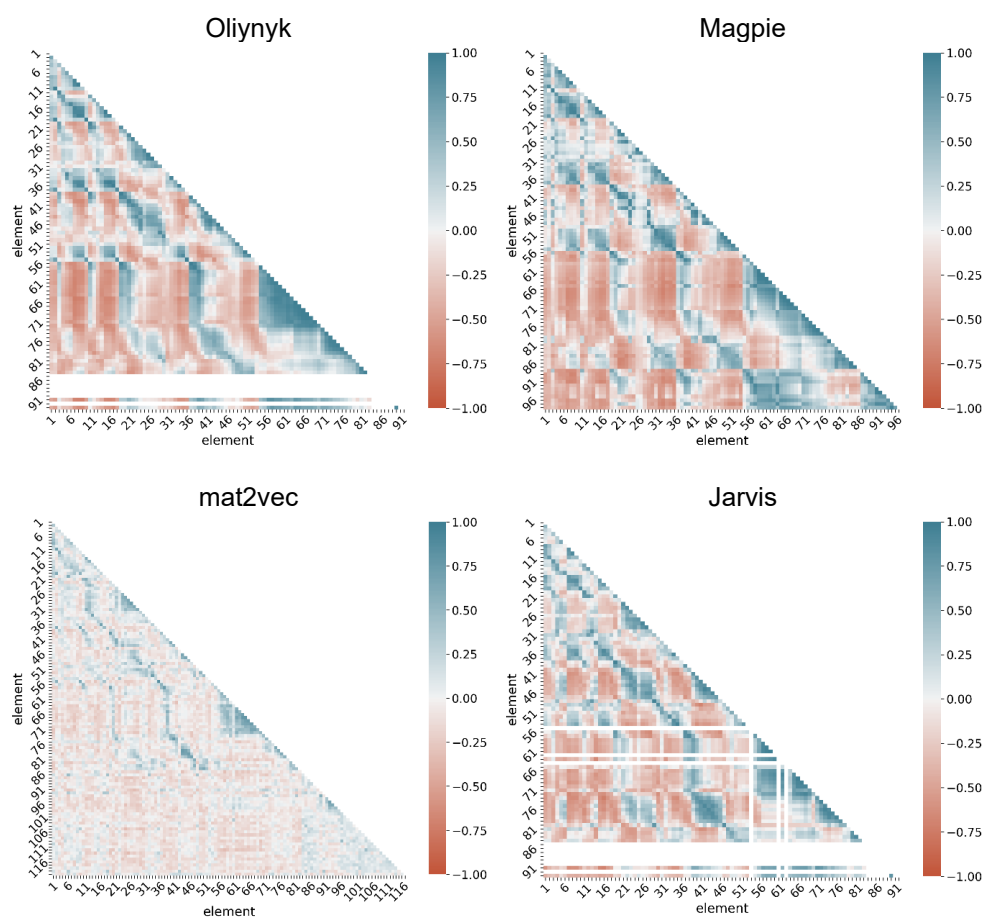


Figure S-1. Uncropped Pearson correlation plots for the element property feature sets Oliynyk, Magpie, mat2vec and Jarvis. The x- and y-axes are labeled with the atomic numbers. Each cell at coordinate (x, y) represents the correlation between the corresponding elements with atomic numbers x and y . Blue represents a high correlation and red represents a high anticorrelation. Empty rows indicate that no element vector is available.

See the **ESM1.zip** file for the full-resolution correlation plots and for interactive HTML versions of the plots (Internet connection required).

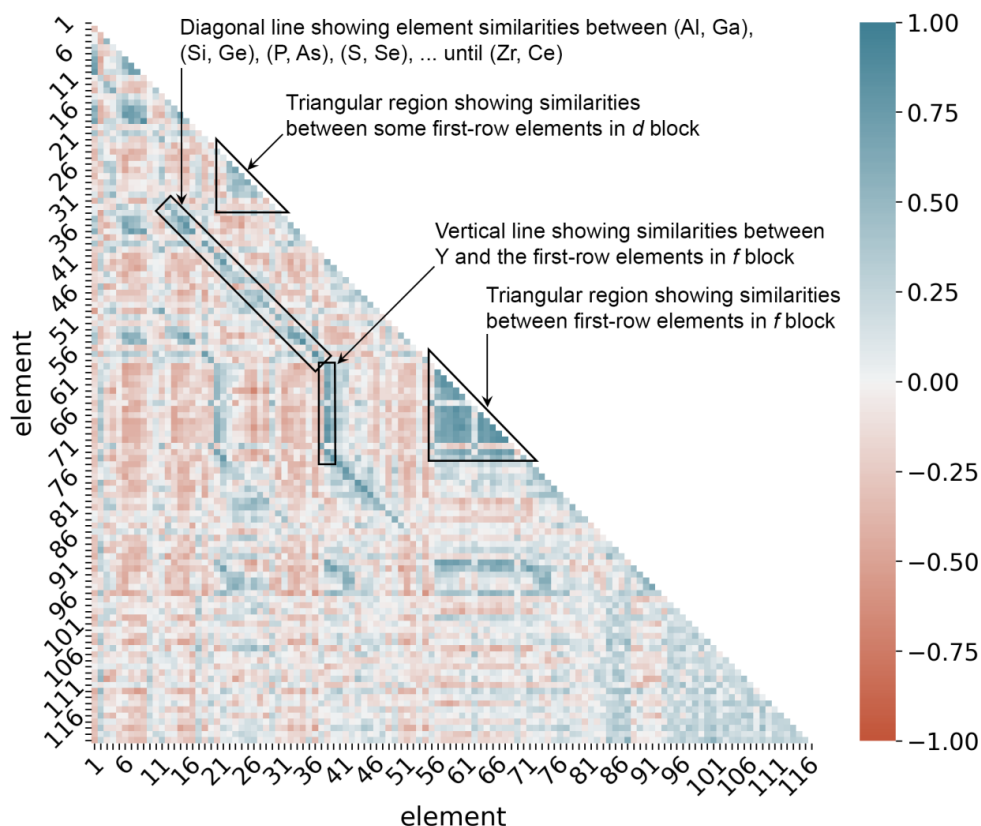


Figure S-2. Annotated heatmap of the Pearson correlation matrix between element vectors extracted from HotCrab for the property OQMD_Bandgap. Here, some regions of interesting similarities between element representations are labeled. These representations are learned entirely from data. The x- and y-axes are labeled with the atomic numbers. Each cell at coordinate (x, y) represents the correlation between the corresponding elements with atomic numbers x and y . Blue represents a high correlation and red represents a high anticorrelation.

See **ESM2.zip** file for more full-resolution correlation plots (including interactive versions) using element vectors extracted from CrabNet and HotCrab.

5.2 PUBLICATION 3: EXPLAINABLEGAP – SI

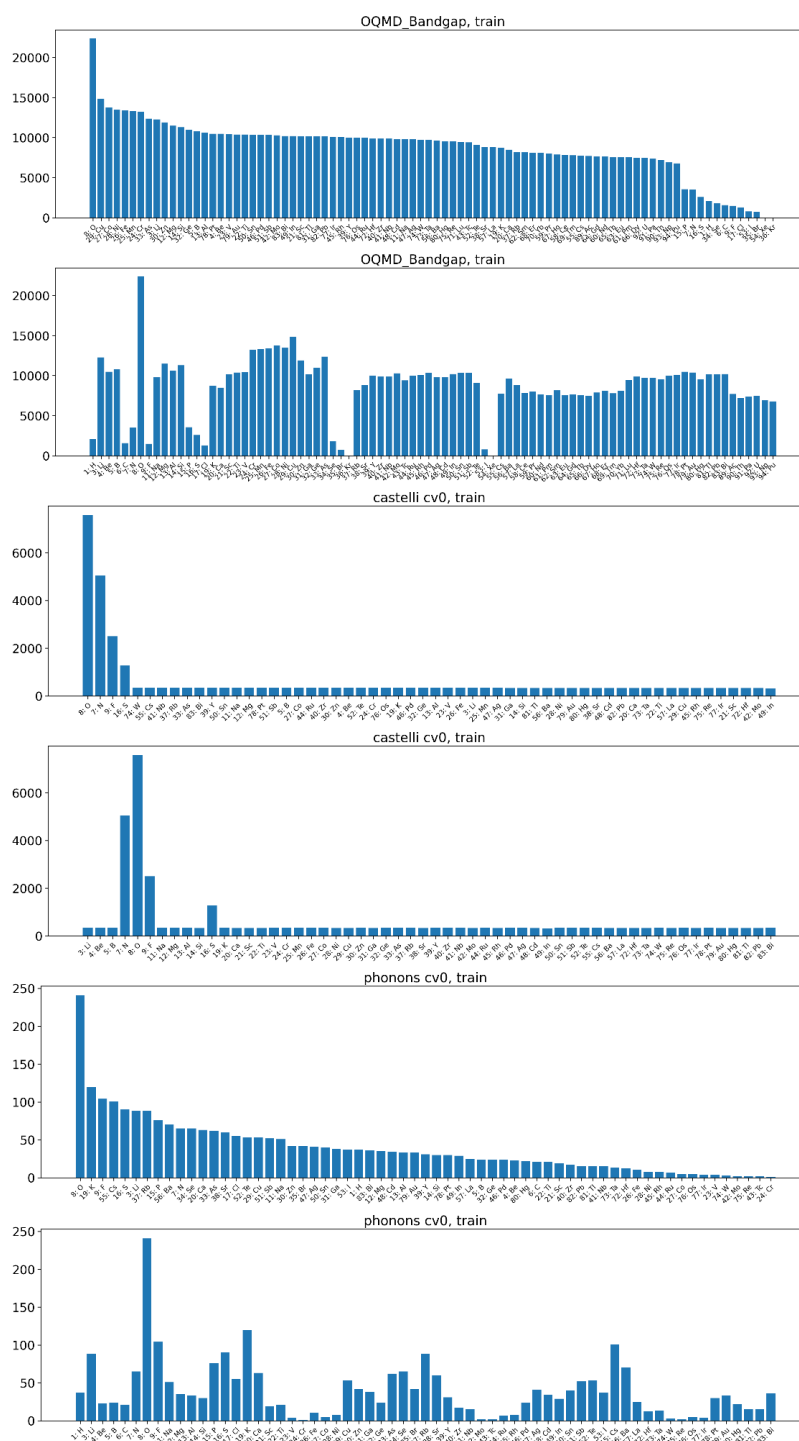


Figure S-3. Example element prevalence plots, sorted by (top) number of occurrences and (bottom) atomic number. See **ESM3.zip** file for more plots.

5 INTERPRETABLE DEEP LEARNING WITH CRABNET

Integrating Materials and Manufacturing Innovation

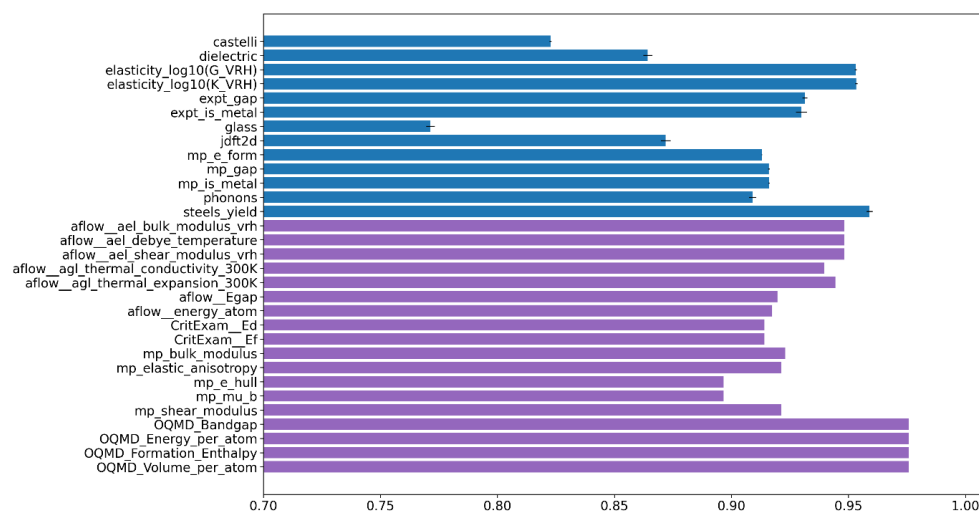


Figure S-4. Shannon equitability indices calculated from the training data splits of the 28 reported datasets in the CrabNet publication. (top) Matbench dataset, (bottom) Extended dataset.

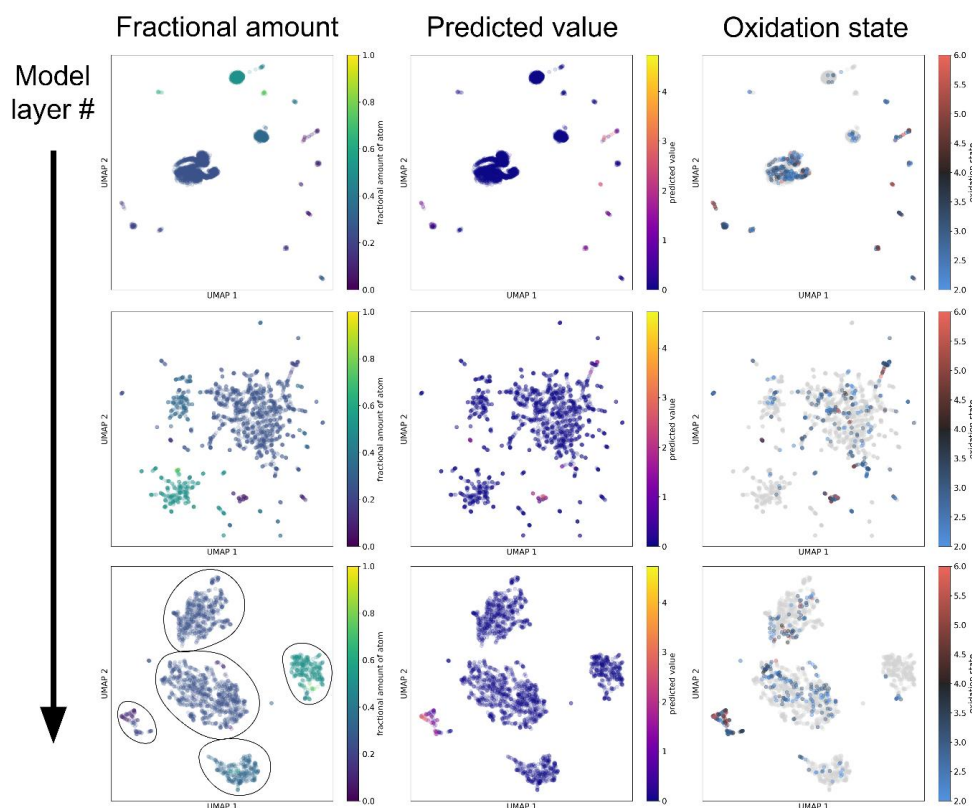


Figure S-5. Vector representations of the chromium element in 2800 different chemical environments and at different layers of the HotCrab model. Each point shows the model-internal representation of the chromium atom, after the information regarding the other atoms in the chemical environment have been introduced via HotCrab through the three attention layers (top row to bottom row). The points are colored by: (left column) the fractional abundance of chromium, (center column) the predicted value of the compound, and (right column) the predicted oxidation state of chromium, where gray points indicate that the oxidation state was unable to be predicted. Five clusters are outlined in the bottom-left plot.

See the **ESM4.zip** file for the individual full-resolution plots and for interactive HTML versions of the plots, for the elements silicon and chromium (Internet connection required).

Supplementary Videos

The attention video files are collected and compressed in a ZIP file (**ESM5.zip**). Please find the files on the figshare website as described on the cover page of this document.

6 Summary and outlook

The **first goal** of this work was achieved through the presentation of clear and concise information about the background of materials informatics, the considerations for the implementation of data-driven methods in materials science, and the demonstration of a typical machine learning project in the form of a project tutorial through Python Jupyter notebooks. Together, the set of guidelines and best practices presented ensures that future publications in the field of materials informatics are reliable, reproducible, and comparable. Furthermore, a selection of materials informatics studies in the literature was highlighted together with an overview of the common materials data repositories, software tools and methodologies used. The methods and protocols suggested will give the interested researcher a solid knowledge foundation and act as a starting point for their first foray into data-driven materials science and materials informatics.

To achieve the **second goal** of this work, a novel, EDM-based featurization technique for the encoding of chemical compositions was introduced, which overcomes several of the limitations with existing featurization techniques. The EDM encodes element identity within the chemical compounds while simultaneously preserving the chemical resolution when featurizing trace elements. Furthermore, a novel deep learning model based on the Transformer self-attention mechanism, named CrabNet, was developed and introduced for the prediction of materials properties. CrabNet was benchmarked against other common deep learning and classical machine learning models on a set of 28 benchmark datasets, and was found to either match or outperform the state of the art in materials property regression tasks.

Crucially, CrabNet and the EDM featurization scheme require only chemical formulae as input data and does not rely on crystal structure information, thus making CrabNet an ideal model for materials discovery tasks. Additionally, the opportunities to explore model interpretability were discussed and previewed. Lastly, the source code and trained model weights of CrabNet are fully open source and made available online. The detailed documentation provided with the code enables interested readers to adapt and

extend the functionalities of CrabNet for their own materials informatics studies, thus promoting further research in the attention-based learning of materials properties.

Last but not least, the **third goal** of this work was achieved through the successful integration of model interpretability methods into CrabNet, further continuing the efforts of turning previously black-box or grey-box models such as CrabNet into fully interpretable models. By extracting the model-internal representations of the featurized elements and compounds as they pass through key CrabNet layers, it was possible to follow the transformations of the data through various stages of modeling and abstraction. Through additional processing of the extracted information, it was shown that CrabNet has successfully learned to represent elements and element relationships for a variety of materials properties, based solely on training data in the form of chemical formulae.

Additionally, it was possible to observe the changing behaviour of individual elements inside compositions as information about their surrounding chemical environment information was incorporated by CrabNet. Furthermore, dense, clusterable visualizations of whole chemical compounds can be visualized and characterized by trends in element stoichiometry, local chemical environment, oxidation state of the elements, and bond behaviour of the compounds. Lastly, the examination of dataset imbalance and self-attention matrices in the form of attention videos can be used to further improve the modeling process, debug potential errors, and gain additional insights about the chemical phenomena underlying material properties.

With the addition of model interpretability to CrabNet, the findings and intuitions presented in this work may lead to further insightful and interesting research as well as increased trust and adoption of CrabNet in academia and industry.

The results presented in this work also suggest **future research directions** which can be broadly categorized into three thematic topics:

1) *Improved compatibility and comparability of models*

The best practice methods and protocols suggested in this work are a good starting point for researchers looking to begin experimenting with data-driven methods in materials science and materials informatics. The introduction of more advanced methods of data processing, feature engineering, modeling and results verification as well as improvements in the materials informatics ecosystem (such

as materials datasets, repositories, model hosting and sharing platforms) will no doubt further expand the set of best practices available to researchers in the future. Here, care should be taken to ensure that future works retain cross-compatibility and comparability with each other as well as with past works (such as the ones presented here).

2) *Improved models tailored to specific materials tasks*

The composition featurization and modeling techniques introduced in this work with CrabNet have pushed the state of the art in structure-agnostic materials property modeling. Due to its unique model architecture, CrabNet will likely remain relevant and a top-performer for a while, especially for cases where other methods fall short (such as for dopants, small data, and extrapolation tasks). The flexibility of CrabNet’s architecture and the underlying self-attention mechanism allow for the adaptation of the model to suit a variety of different tasks, such as for the classification of materials or for the specific learning of element and compound interactions (*e.g.*, in catalysis).

Furthermore, integration of additional or other input data such as crystal structure or molecular connectivity may allow CrabNet to expand and improve its predictive power in specific tasks. Additionally, the improved ability to featurize compositions suggests the prospect of modeling even complexer materials properties such as those resulting from multi-phase or disordered systems. This development will also depend on the coordinated improvements in the quality and quantity of specialized materials datasets in data repositories.

3) *Improved model interpretability techniques*

Intrinsic model interpretability methods provide crucial information in the search of the novel materials of tomorrow. The techniques shown in this current work to integrate and expand model interpretability in CrabNet demonstrate that it is possible to unlock black-box modeling techniques in materials science with deliberate and prudent model design choices. The findings in this work will likely lead to further improvements of model interpretability techniques in CrabNet and other models alike, with future studies developing and examining other model architectures to better understand modeling and decision-making processes.

6 SUMMARY AND OUTLOOK

The application of data-driven methods has the potential to address and provide insights to many existing materials science challenges. In this regard, the results of this work will lead to a stronger understanding of materials modeling and a quicker improvement of the crucial tools used in materials informatics. With better models and established best practices, more researchers and industrial practitioners will adopt materials informatics as part of their materials design and discovery processes. Undoubtedly, this will bring about additional insights into the chemical phenomena underlying materials properties, and, perhaps one day, *engineered serendipity*.

Computers designing compounds; materials discovering materials—imagine that!



7 References

- [1] A. Y.-T. Wang, R. J. Murdock, S. K. Kauwe, A. O. Oliynyk, A. Gurlo, J. Brgoch, K. A. Persson, and T. D. Sparks. “Machine Learning for Materials Scientists: An Introductory Guide toward Best Practices”. *Chemistry of Materials*, 32 (12) (2020), 4954–4965. DOI: 10.1021/acs.chemmater.0c01907 (cit. on pp. 1, 19).
- [2] A. Y.-T. Wang, S. K. Kauwe, R. J. Murdock, and T. D. Sparks. “Compositionally restricted attention-based network for materials property predictions”. *npj Computational Materials*, 7 (2021), 77. DOI: 10.1038/s41524-021-00545-1 (cit. on pp. 1, 119).
- [3] A. Y.-T. Wang, M. S. Mahmoud, M. Czasny, and A. Gurlo. “CrabNet for Explainable Deep Learning in Materials Science: Bridging the Gap Between Academia and Industry”. *Integrating Materials and Manufacturing Innovation*, (2022). DOI: 10.1007/s40192-021-00247-y (cit. on pp. 1, 139).
- [4] R. J. Murdock, S. K. Kauwe, A. Y.-T. Wang, and T. D. Sparks. “Is Domain Knowledge Necessary for Machine Learning Materials Properties?” *Integrating Materials and Manufacturing Innovation*, 9 (3) (2020), 221–227. DOI: 10.1007/s40192-020-00179-z (cit. on pp. 1, 113, 114).
- [5] T. A. Edison. “The Beginnings of the Incandescent Lamp”. *Scientific American Supplement*, 57 (1480) (1904), 23711–23712. URL: https://archive.org/details/sim_scientific-american-supplement_1904-05-14_57_1480 (cit. on p. 7).
- [6] T. A. Edison. “Electric lamp”. US 223898 A. 1880 (cit. on p. 7).
- [7] M. F. Bosenman. “Serendipity and Scientific Discovery”. *The Journal of Creative Behavior*, 22 (2) (1988), 132–138. ISSN: 0022-0175. DOI: 10.1002/j.2162-6057.1988.tb00674.x (cit. on p. 7).
- [8] R. W. Cahn. “Challenge, response and serendipity in the design of materials”. *Bulletin of Materials Science*, 17 (7) (1994), 1369–1378. ISSN: 0250-4707. DOI: 10.1007/BF02747234 (cit. on p. 7).
- [9] K. Rajan. “Materials Informatics”. *Materials Today*, 8 (10) (2005), 38–45. ISSN: 1369-7021. DOI: 10.1016/S1369-7021(05)71123-8 (cit. on pp. 7, 10, 13).
- [10] M. P. E. Cunha. “Serendipity: Why Some Organizations are Luckier than Others”. *SSRN Electronic Journal*, (2005). DOI: 10.2139/ssrn.882782 (cit. on p. 7).
- [11] R. K. Merton and E. G. Barber. *The travels and adventures of serendipity: A study in sociological semantics and the sociology of science*. Princeton paperbacks. Princeton, NJ: Princeton University Press, 2006. ISBN: 978-0-691-12630-2 (cit. on p. 7).

7 REFERENCES

- [12] R. Campa. "Making Science by Serendipity: A review of Robert K. Merton and Elinor Barber's 'The Travels and Adventures of Serendipity'". *Journal of Evolution and Technology*, 17 (1) (2008), 75–83. URL: <https://jetpress.org/v17/campa.htm> (cit. on p. 7).
- [13] M. de Rond. "The structure of serendipity". *Culture and Organization*, 20 (5) (2014), 342–358. ISSN: 1475-9551. DOI: 10.1080/14759551.2014.967451 (cit. on p. 7).
- [14] S. Copeland. "On serendipity in science: Discovery at the intersection of chance and wisdom". *Synthese*, 163 (April) (2017), 30. ISSN: 0039-7857. DOI: 10.1007/s11229-017-1544-3 (cit. on p. 7).
- [15] Nature Editorial. "The serendipity test". *Nature*, 554 (7690) (2018), 5. ISSN: 0028-0836. DOI: 10.1038/d41586-018-01405-7 (cit. on p. 7).
- [16] O. Yaqub. "Serendipity: Towards a taxonomy and a theory". *Research Policy*, 47 (1) (2018), 169–179. ISSN: 0048-7333. DOI: 10.1016/j.respol.2017.10.007 (cit. on p. 7).
- [17] C. Busch. *The Serendipity Mindset: The Art and Science of Creating Good Luck*. London: Penguin Books, 2020. ISBN: 978-0-241-40209-2 (cit. on p. 7).
- [18] National Science Foundation. *Leading Engineers and Scientists Identify Advances That Could Improve Quality of Life Around the World: News Release 08-028*. 2008. URL: https://www.nsf.gov/news/news_summ.jsp?cntn_id=111158 (last access 09/24/2021) (cit. on p. 8).
- [19] J. Graser, S. K. Kauwe, and T. D. Sparks. "Machine Learning and Energy Minimization Approaches for Crystal Structure Predictions: A Review and New Horizons". *Chemistry of Materials*, 30 (11) (2018), 3601–3612. DOI: 10.1021/acs.chemmater.7b05304 (cit. on pp. 8, 10, 12, 15, 113).
- [20] A. Walsh. "Inorganic materials: The quest for new functionality". *Nature chemistry*, 7 (4) (2015), 274–275. DOI: 10.1038/nchem.2213 (cit. on p. 9).
- [21] T. D. Sparks, M. W. Gaultois, A. O. Oliynyk, J. Brgoch, and B. Meredig. "Data mining our way to the next generation of thermoelectrics". *Scripta Materialia*, 111 (2016), 10–15. ISSN: 1359-6462. DOI: 10.1016/j.scriptamat.2015.04.026 (cit. on pp. 10, 15).
- [22] T. Bligaard, G. H. Jóhannesson, A. V. Ruban, H. L. Skriver, K. W. Jacobsen, and J. K. Nørskov. "Pareto-optimal alloys". *Applied Physics Letters*, 83 (22) (2003), 4527–4529. ISSN: 0003-6951. DOI: 10.1063/1.1631051 (cit. on p. 10).
- [23] G. J. Mulholland and S. P. Paradiso. "Perspective: Materials informatics across the product lifecycle: Selection, manufacturing, and certification". *APL Materials*, 4 (5) (2016), 053207. DOI: 10.1063/1.4945422 (cit. on p. 10).
- [24] P. Raccuglia et al. "Machine-learning-assisted materials discovery using failed experiments". *Nature*, 533 (7601) (2016), 73–76. ISSN: 0028-0836. DOI: 10.1038/nature17439 (cit. on p. 10).

- [25] G. R. Schleder, A. C. M. Padilha, C. M. Acosta, M. Costa, and A. Fazzio. "From DFT to machine learning: recent approaches to materials science—a review". *Journal of Physics: Materials*, 2 (3) (2019), 032001. DOI: 10.1088/2515-7639/ab084b (cit. on p. 10).
- [26] C. P. Gomes, B. Selman, and J. M. Gregoire. "Artificial intelligence for materials discovery". *MRS Bulletin*, 44 (7) (2019), 538–544. ISSN: 0883-7694. DOI: 10.1557/mrs.2019.158 (cit. on pp. 10, 15, 16).
- [27] T. D. Sparks, S. K. Kauwe, M. E. Parry, A. Mansouri Tehrani, and J. Bragoch. "Machine Learning for Structural Materials". *Annual Review of Materials Research*, 50 (1) (2020). ISSN: 1531-7331. DOI: 10.1146/annurev-matsci-110519-094700 (cit. on pp. 10, 16).
- [28] D. J. d. S. Price. *Science Since Babylon*. Enl. ed. New Haven: Yale University Press, 1975. ISBN: 978-0-300-01798-4 (cit. on p. 11).
- [29] L. Bornmann and R. Mutz. "Growth rates of modern science: A bibliometric analysis based on the number of publications and cited references". *Journal of the Association for Information Science and Technology*, 66 (11) (2015), 2215–2222. ISSN: 2330-1635. DOI: 10.1002/asi.23329 (cit. on p. 11).
- [30] National Science Foundation. *Science and Engineering Indicators 2010*. Arlington, VA, 2010. URL: <https://wayback.archive-it.org/5902/20150627170008/http://www.nsf.gov/statistics/seind10/start.htm> (last access 09/23/2021) (cit. on p. 11).
- [31] T. D. Sparks. 2. *What is Materials Informatics?* Online, 2021. URL: <https://www.youtube.com/watch?v=65vCdOibwhQ> (last access 11/22/2021) (cit. on p. 11).
- [32] A. Agrawal and A. Choudhary. "Perspective: Materials informatics and big data: Realization of the "fourth paradigm" of science in materials science". *APL Materials*, 4 (5) (2016), 053208. DOI: 10.1063/1.4946894 (cit. on pp. 13, 15).
- [33] A. J. G. Hey, S. Tansley, and K. M. Tolle. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Redmond, WA: Microsoft Research, 2009. ISBN: 978-0-9825442-0-4 (cit. on p. 13).
- [34] K. T. Butler, D. W. Davies, H. Cartwright, O. Isayev, and A. Walsh. "Machine learning for molecular and materials science". *Nature*, 559 (7715) (2018), 547–555. ISSN: 0028-0836. DOI: 10.1038/s41586-018-0337-2 (cit. on pp. 13, 16).
- [35] G. Ceder. "Predicting Properties from Scratch". *Science*, 280 (5366) (1998), 1099–1100. ISSN: 0036-8075. DOI: 10.1126/science.280.5366.1099 (cit. on p. 13).
- [36] S. R. Kalidindi and M. de Graef. "Materials Data Science: Current Status and Future Outlook". *Annual Review of Materials Research*, 45 (1) (2015), 171–193. ISSN: 1531-7331. DOI: 10.1146/annurev-matsci-070214-020844 (cit. on pp. 13, 16).
- [37] R. Ramprasad, R. Batra, G. Pilania, A. Mannodi-Kanakkithodi, and C. Kim. "Machine learning in materials informatics: Recent applications and prospects". *npj Computational Materials*, 3 (1) (2017), 60. DOI: 10.1038/s41524-017-0056-5 (cit. on pp. 13, 16, 113).

7 REFERENCES

- [38] B. Meredig. "Industrial materials informatics: Analyzing large-scale data to solve applied problems in R&D, manufacturing, and supply chain". *Current Opinion in Solid State & Materials Science*, 21 (3) (2017), 159–166. ISSN: 1359-0286. DOI: 10.1016/j.cossms.2017.01.003 (cit. on p. 13).
- [39] R. Seshadri and T. D. Sparks. "Perspective: Interactive material property databases through aggregation of literature data". *APL Materials*, 4 (5) (2016), 053206. DOI: 10.1063/1.4944682 (cit. on p. 13).
- [40] T. Lookman, F. J. Alexander, and K. Rajan. *Information Science for Materials Discovery and Design*. 225. Springer Series in Materials Science. Cham, Switzerland: Springer, 2016. ISBN: 978-3-319-23871-5. DOI: 10.1007/978-3-319-23871-5 (cit. on pp. 13, 16).
- [41] K. Rajan. "Materials Informatics: The Materials "Gene" and Big Data". *Annual Review of Materials Research*, 45 (1) (2015), 153–169. ISSN: 1531-7331. DOI: 10.1146/annurev-matsci-070214-021132 (cit. on pp. 13, 16).
- [42] J. Schmidt, M. R. G. Marques, S. Botti, and M. A. L. Marques. "Recent advances and applications of machine learning in solid-state materials science". *npj Computational Materials*, 5 (1) (2019), 83. DOI: 10.1038/s41524-019-0221-0 (cit. on pp. 13, 16).
- [43] J. Li, K. Lim, H. Yang, Z. Ren, S. Raghavan, P.-Y. Chen, T. Buonassisi, and X. Wang. "AI Applications through the Whole Life Cycle of Material Discovery". *Matter*, 3 (2) (2020), 393–432. ISSN: 2590-2393. DOI: 10.1016/j.matt.2020.06.011 (cit. on p. 13).
- [44] National Institute of Standards and Technology (NIST). *NIST Materials Genome Initiative*. 2011. URL: <https://mgi.nist.gov/> (last access 08/05/2021) (cit. on p. 13).
- [45] C. M. Bishop. *Pattern recognition and machine learning*. Information science and statistics. New York, NY: Springer, 2006. ISBN: 978-0387-31073-2 (cit. on p. 14).
- [46] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive computation and machine learning series. Cambridge, MA: The MIT Press, 2016. ISBN: 9780262035613 (cit. on p. 14).
- [47] P. Mehta, C.-H. Wang, A. G. R. Day, C. Richardson, M. Bukov, C. K. Fisher, and D. J. Schwab. "A high-bias, low-variance introduction to Machine Learning for physicists". *Physics reports*, 810 (2019), 1–124. ISSN: 0370-1573. DOI: 10.1016/j.physrep.2019.03.001 (cit. on p. 14).
- [48] M. T. Lusk and A. E. Mattsson. "High-performance computing for materials design to advance energy science". *MRS Bulletin*, 36 (3) (2011), 169–174. ISSN: 0883-7694. DOI: 10.1557/mrs.2011.30 (cit. on p. 14).
- [49] J. Lee, A. Seko, K. Shitara, K. Nakayama, and I. Tanaka. "Prediction model of band gap for inorganic compounds by combination of density functional theory calculations and machine learning techniques". *Physical Review B*, 93 (11) (2016), 115104. ISSN: 1098-0121. DOI: 10.1103/PhysRevB.93.115104 (cit. on p. 14).

- [50] C. Chen, Y. Zuo, W. Ye, X. Li, Z. Deng, and S. P. Ong. "A Critical Review of Machine Learning of Energy Materials". *Advanced Energy Materials*, 10 (8) (2020), 1903242. ISSN: 1614-6832. DOI: 10.1002/aenm.201903242 (cit. on pp. 14, 116).
- [51] T. D. Sparks. *Accuracy, Uncertainty, Inspectability: The Benefit of Composition-restricted Attention-based Network*. Online, 2021. URL: <https://www.youtube.com/watch?v=KtUstqOL33w> (cit. on p. 15).
- [52] L. Ward, A. Agrawal, A. Choudhary, and C. Wolverton. "A general-purpose machine learning framework for predicting properties of inorganic materials". *npj Computational Materials*, 2 (1) (2016), 16028. DOI: 10.1038/npjcompumats.2016.28 (cit. on pp. 15, 113).
- [53] L. Wei, X. Xu, Gurudayal, J. Bullock, and J. W. Ager. "Machine Learning Optimization of p-Type Transparent Conducting Films". *Chemistry of Materials*, 31 (18) (2019), 7340–7350. DOI: 10.1021/acs.chemmater.9b01953 (cit. on p. 15).
- [54] D. W. Davies, K. T. Butler, and A. Walsh. "Data-Driven Discovery of Photoactive Quaternary Oxides Using First-Principles Machine Learning". *Chemistry of Materials*, 31 (18) (2019), 7221–7230. DOI: 10.1021/acs.chemmater.9b01519 (cit. on p. 15).
- [55] A. D. Sendek, Q. Yang, E. D. Cubuk, K.-A. N. Duerloo, Y. Cui, and E. J. Reed. "Holistic computational structure screening of more than 12000 candidates for solid lithium-ion conductor materials". *Energy & Environmental Science*, 10 (1) (2017), 306–320. ISSN: 1754-5692. DOI: 10.1039/C6EE02697D (cit. on p. 15).
- [56] Z. Ahmad, T. Xie, C. Maheshwari, J. C. Grossman, and V. Viswanathan. "Machine Learning Enabled Computational Screening of Inorganic Solid Electrolytes for Suppression of Dendrite Formation in Lithium Metal Anodes". *ACS Central Science*, 4 (8) (2018), 996–1006. ISSN: 2374-7943. DOI: 10.1021/acscentsci.8b00229 (cit. on p. 15).
- [57] A. D. Sendek, E. D. Cubuk, E. R. Antoniuk, G. Cheon, Y. Cui, and E. J. Reed. "Machine Learning-Assisted Discovery of Solid Li-Ion Conducting Materials". *Chemistry of Materials*, 31 (2) (2019), 342–352. DOI: 10.1021/acs.chemmater.8b03272 (cit. on p. 15).
- [58] N. S. Bobbitt and R. Q. Snurr. "Molecular modelling and machine learning for high-throughput screening of metal-organic frameworks for hydrogen storage". *Molecular Simulation*, 45 (14-15) (2019), 1069–1081. ISSN: 0892-7022. DOI: 10.1080/08927022.2019.1597271 (cit. on p. 15).
- [59] G. H. Gu, J. Noh, I. Kim, and Y. Jung. "Machine learning for renewable energy materials". *Journal of Materials Chemistry A*, 7 (29) (2019), 17096–17117. ISSN: 2050-7488. DOI: 10.1039/C9TA02356A (cit. on p. 15).
- [60] Z. W. Seh, J. Kibsgaard, C. F. Dickens, I. Chorkendorff, J. K. Nørskov, and T. F. Jaramillo. "Combining theory and experiment in electrocatalysis: Insights into materials design". *Science*, 355 (6321) (2017), eaad4998. ISSN: 0036-8075. DOI: 10.1126/science.aad4998 (cit. on p. 15).
- [61] Z. W. Ulissi, A. J. Medford, T. Bligaard, and J. K. Nørskov. "To address surface reaction network complexity using scaling relations machine learning and DFT calculations".

7 REFERENCES

- Nature Communications*, 8 (2017), 14621. ISSN: 2041-1723. DOI: 10.1038/ncomms14621 (cit. on p. 15).
- [62] J. R. Kitchin. “Machine learning in catalysis”. *Nature Catalysis*, 1 (4) (2018), 230–232. DOI: 10.1038/s41929-018-0056-y (cit. on p. 15).
- [63] M. H. Hansen, J. A. G. Torres, P. C. Jennings, Z. Wang, J. R. Boes, O. G. Mamun, and T. Bligaard. *An Atomistic Machine Learning Package for Surface Science and Catalysis*. arXiv. 2019. URL: <http://arxiv.org/pdf/1904.00904v1> (last access 05/05/2020) (cit. on p. 15).
- [64] P. Schlexer Lamoureux, K. T. Winther, J. A. Garrido Torres, V. Streibel, M. Zhao, M. Bajdich, F. Abild-Pedersen, and T. Bligaard. “Machine Learning for Computational Heterogeneous Catalysis”. *ChemCatChem*, 11 (16) (2019), 3581–3601. ISSN: 1867-3880. DOI: 10.1002/cctc.201900595 (cit. on p. 15).
- [65] H. Masood, C. Y. Toe, W. Y. Teoh, V. Sethu, and R. Amal. “Machine Learning for Accelerated Discovery of Solar Photocatalysts”. *ACS Catalysis*, 9 (12) (2019), 11774–11787. ISSN: 2155-5435. DOI: 10.1021/acscatal.9b02531 (cit. on p. 15).
- [66] A. K. Singh, J. H. Montoya, J. M. Gregoire, and K. A. Persson. “Robust and synthesizable photocatalysts for CO₂ reduction: a data-driven materials discovery”. *Nature Communications*, 10 (1) (2019), 443. ISSN: 2041-1723. DOI: 10.1038/s41467-019-08356-1 (cit. on p. 15).
- [67] M. W. Gaultois, T. D. Sparks, C. K. H. Borg, R. Seshadri, W. D. Bonificio, and D. R. Clarke. “Data-Driven Review of Thermoelectric Materials: Performance and Resource Considerations”. *Chemistry of Materials*, 25 (15) (2013), 2911–2920. DOI: 10.1021/cm400893e (cit. on p. 15).
- [68] M. W. Gaultois, A. O. Oliynyk, A. Mar, T. D. Sparks, G. J. Mulholland, and B. Meredig. “Perspective: Web-based machine learning models for real-time screening of thermoelectric materials properties”. *APL Materials*, 4 (5) (2016), 053213. DOI: 10.1063/1.4952607 (cit. on pp. 15, 113).
- [69] A. O. Oliynyk, E. Antono, T. D. Sparks, L. Ghadbeigi, M. W. Gaultois, B. Meredig, and A. Mar. “High-Throughput Machine-Learning-Driven Synthesis of Full-Heusler Compounds”. *Chemistry of Materials*, 28 (20) (2016), 7324–7331. DOI: 10.1021/acs.chemmater.6b02724 (cit. on pp. 15, 113).
- [70] V. Tshitoyan et al. “Unsupervised word embeddings capture latent knowledge from materials science literature”. *Nature*, 571 (7763) (2019), 95–98. ISSN: 0028-0836. DOI: 10.1038/s41586-019-1335-8 (cit. on pp. 15, 113).
- [71] V. Stanev, C. Oses, A. G. Kusne, E. Rodriguez, J. Paglione, S. Curtarolo, and I. Takeuchi. “Machine learning modeling of superconducting critical temperature”. *npj Computational Materials*, 4 (1) (2018), 29. DOI: 10.1038/s41524-018-0085-8 (cit. on p. 15).
- [72] B. Meredig et al. “Can machine learning identify the next high-temperature superconductor? Examining extrapolation performance for materials discovery”. *Molecular Systems Design & Engineering*, 3 (5) (2018), 819–825. DOI: 10.1039/C8ME00012C (cit. on p. 15).

- [73] T. Konno, H. Kurokawa, F. Nabeshima, Y. Sakishita, R. Ogawa, I. Hosako, and A. Maeda. *Deep Learning Model for Finding New Superconductors*. arXiv. 2018. URL: <http://arxiv.org/pdf/1812.01995v3> (last access 05/05/2020) (cit. on p. 15).
- [74] K. Hamidieh. "A data-driven statistical model for predicting the critical temperature of a superconductor". *Computational Materials Science*, 154 (2018), 346–354. ISSN: 0927-0256. DOI: 10.1016/j.commatsci.2018.07.052 (cit. on p. 15).
- [75] Y. Dan, R. Dong, Z. Cao, X. Li, C. Niu, S. Li, and J. Hu. "Computational Prediction of Critical Temperatures of Superconductors Based on Convolutional Gradient Boosting Decision Trees". *IEEE Access*, 8 (2020), 57868–57878. DOI: 10.1109/ACCESS.2020.2981874 (cit. on p. 15).
- [76] K. Matsumoto and T. Horide. "An acceleration search method of higher T_c superconductors by a machine learning algorithm". *Applied Physics Express*, 12 (7) (2019), 073003. ISSN: 1882-0778. DOI: 10.7567/1882-0786/ab2922 (cit. on p. 15).
- [77] B. Roter and S. V. Dordevic. *Predicting new superconductors and their critical temperatures using unsupervised machine learning*. arXiv. 2020. URL: <http://arxiv.org/pdf/2002.07266v1> (last access 05/05/2020) (cit. on p. 15).
- [78] C. Wen et al. "Machine learning assisted design of high entropy alloys with desired property". *Acta Materialia*, 170 (2019), 109–117. ISSN: 1359-6454. DOI: 10.1016/j.actamat.2019.03.010 (cit. on p. 15).
- [79] Y.-J. Chang, C.-Y. Jui, W.-J. Lee, and A.-C. Yeh. "Prediction of the Composition and Hardness of High-Entropy Alloys by Machine Learning". *JOM*, 71 (2019), 3433–3442. ISSN: 1047-4838. DOI: 10.1007/s11837-019-03704-4 (cit. on p. 15).
- [80] F. Ren, L. Ward, T. Williams, K. J. Laws, C. Wolverton, J. Hattrick-Simpers, and A. Mehta. "Accelerated discovery of metallic glasses through iteration of machine learning and high-throughput experiments". *Science Advances*, 4 (4) (2018), eaaq1566. DOI: 10.1126/sciadv.aaq1566 (cit. on p. 15).
- [81] L. Ward, S. C. O’Keeffe, J. Stevick, G. R. Jelbert, M. Aykol, and C. Wolverton. "A machine learning approach for engineering bulk metallic glass alloys". *Acta Materialia*, 159 (2018), 102–111. ISSN: 1359-6454. DOI: 10.1016/j.actamat.2018.08.002 (cit. on p. 15).
- [82] T. Mueller, A. G. Kusne, and R. Ramprasad. "Machine Learning in Materials Science". *Reviews in Computational Chemistry*. Ed. by A. L. Parrill and K. B. Lipkowitz. 1. Reviews in Computational Chemistry. Hoboken, NJ: John Wiley & Sons, Inc., 2016, 186–273. ISBN: 978-1-119-14873-9. DOI: 10.1002/9781119148739.ch4 (cit. on p. 16).
- [83] Y. Liu, T. Zhao, W. Ju, and S. Shi. "Materials discovery and design using machine learning". *Journal of Materiomics*, 3 (3) (2017), 159–177. ISSN: 2352-8478. DOI: 10.1016/j.jmat.2017.08.002 (cit. on p. 16).
- [84] P. Gorai, V. Stevanović, and E. S. Toberer. "Computationally guided discovery of thermoelectric materials". *Nature Reviews Materials*, 2 (9) (2017), 17053. ISSN: 2058-8437. DOI: 10.1038/natrevmats.2017.53 (cit. on p. 16).

7 REFERENCES

- [85] D. P. Tabor et al. "Accelerating the discovery of materials for clean energy in the era of smart automation". *Nature Reviews Materials*, 3 (5) (2018), 5–20. ISSN: 2058-8437. DOI: 10.1038/s41578-018-0005-z (cit. on p. 16).
- [86] S. Ramakrishna et al. "Materials Informatics". *Journal of Intelligent Manufacturing*, 4 (5) (2018), 053208. ISSN: 0956-5515. DOI: 10.1007/s10845-018-1392-0 (cit. on p. 16).
- [87] J. M. Rickman, T. Lookman, and S. V. Kalinin. "Materials informatics: From the atomic-level to the continuum". *Acta Materialia*, 168 (2019), 473–510. ISSN: 1359-6454. DOI: 10.1016/j.actamat.2019.01.051 (cit. on p. 16).
- [88] O. Isayev, A. Tropsha, and S. Curtarolo, eds. *Materials Informatics: Methods, Tools, and Applications*. Wiley, 2019. ISBN: 978-3-527-34121-4. DOI: 10.1002/9783527802265 (cit. on p. 16).
- [89] S. P. Ong. "Accelerating materials science with high-throughput computations and machine learning". *Computational Materials Science*, 161 (2019), 143–150. ISSN: 0927-0256. DOI: 10.1016/j.commatsci.2019.01.013 (cit. on p. 16).
- [90] B. Meredig. "Five High-Impact Research Areas in Machine Learning for Materials Science". *Chemistry of Materials*, 31 (23) (2019), 9579–9581. DOI: 10.1021/acs.chemmater.9b04078 (cit. on p. 16).
- [91] R. K. Vasudevan et al. "Materials science in the artificial intelligence age: high-throughput library generation, machine learning, and a pathway from correlations to the underpinning physics". *MRS Communications*, 9 (03) (2019), 821–838. ISSN: 2159-6859. DOI: 10.1557/mrc.2019.95 (cit. on p. 16).
- [92] H. S. Stein and J. M. Gregoire. "Progress and prospects for accelerating materials science with automated and autonomous workflows". *Chemical Science*, 10 (42) (2019), 9640–9649. ISSN: 2041-6520. DOI: 10.1039/C9SC03766G (cit. on p. 16).
- [93] D. Morgan and R. Jacobs. "Opportunities and Challenges for Machine Learning in Materials Science". *Annual Review of Materials Research*, 50 (1) (2020), 71–103. ISSN: 1531-7331. DOI: 10.1146/annurev-matsci-070218-010015 (cit. on p. 16).
- [94] B. DeCost, J. R. Hattrick-Simpers, Z. Trautt, A. G. Kusne, E. Campo, and M. L. Green. "Scientific AI in materials science: a path to a sustainable and scalable paradigm". *Machine Learning: Science and Technology*, (2020). DOI: 10.1088/2632-2153/ab9a20 (cit. on p. 16).
- [95] C. L. Zitnick et al. *An Introduction to Electrocatalyst Design using Machine Learning for Renewable Energy Storage*. arXiv. Online, 2020. URL: <http://arxiv.org/pdf/2010.09435v1> (cit. on p. 16).
- [96] G. Pilania. "Machine learning in materials science: From explainable predictions to autonomous design". *Computational Materials Science*, 193 (2021), 110360. ISSN: 0927-0256. DOI: 10.1016/j.commatsci.2021.110360 (cit. on p. 16).
- [97] A. S. Barnard. "Best Practice Leads to the Best Materials Informatics". *Matter*, 3 (1) (2020), 22–23. ISSN: 2590-2393. DOI: 10.1016/j.matt.2020.06.003 (cit. on p. 17).

- [98] G. Pilania, A. Mannodi-Kanakkithodi, B. P. Uberuaga, R. Ramprasad, J. E. Gubernatis, and T. Lookman. "Machine learning bandgaps of double perovskites". *Scientific Reports*, 6 (2016), 19375. ISSN: 2045-2322. DOI: 10.1038/srep19375 (cit. on p. 113).
- [99] A. Dunn, Q. Wang, A. Ganose, D. Dopp, and A. Jain. "Benchmarking materials property prediction methods: the Matbench test set and Automatminer reference algorithm". *npj Computational Materials*, 6 (1) (2020), 138. DOI: 10.1038/s41524-020-00406-3 (cit. on p. 113).
- [100] S. K. Kauwe, J. Graser, R. J. Murdock, and T. D. Sparks. "Can machine learning find extraordinary materials?" *Computational Materials Science*, 174 (2020), 109498. ISSN: 0927-0256. DOI: 10.1016/j.commatsci.2019.109498 (cit. on p. 113).
- [101] A. Mansouri Tehrani et al. "Machine Learning Directed Search for Ultraincompressible, Superhard Materials". *Journal of the American Chemical Society*, 140 (31) (2018), 9844–9853. ISSN: 0002-7863. DOI: 10.1021/jacs.8b02717 (cit. on p. 113).
- [102] B. Meredig et al. "Combinatorial screening for new materials in unconstrained composition space with machine learning". *Physical Review B*, 89 (9) (2014), 094104. ISSN: 1098-0121. DOI: 10.1103/PhysRevB.89.094104 (cit. on pp. 113, 116).
- [103] Y. Zhuo, A. Mansouri Tehrani, and J. Brgoch. "Predicting the Band Gaps of Inorganic Solids by Machine Learning". *The Journal of Physical Chemistry Letters*, 9 (7) (2018), 1668–1673. ISSN: 1948-7185. DOI: 10.1021/acs.jpclett.8b00124 (cit. on p. 113).
- [104] V. Venkatraman. "The utility of composition-based machine learning models for band gap prediction". *Computational Materials Science*, 197 (2021), 110637. ISSN: 0927-0256. DOI: 10.1016/j.commatsci.2021.110637 (cit. on pp. 113, 114).
- [105] K. Choudhary, B. DeCost, and F. Tavazza. "Machine learning with force-field-inspired descriptors for materials: Fast screening and mapping energy landscape". *Physical Review Materials*, 2 (8) (2018), 083801. DOI: 10.1103/PhysRevMaterials.2.083801 (cit. on p. 113).
- [106] Q. Zhou, P. Tang, S. Liu, J. Pan, Q. Yan, and S.-C. Zhang. "Learning atoms for materials discovery". *Proceedings of the National Academy of Sciences*, 115 (28) (2018), E6411–E6417. ISSN: 0027-8424. DOI: 10.1073/pnas.1801181115 (cit. on p. 113).
- [107] D. Jha, L. Ward, A. Paul, W.-K. Liao, A. Choudhary, C. Wolverton, and A. Agrawal. "ElemNet: Deep Learning the Chemistry of Materials From Only Elemental Composition". *Scientific Reports*, 8 (1) (2018), 17593. ISSN: 2045-2322. DOI: 10.1038/s41598-018-35934-y (cit. on pp. 113, 116).
- [108] W. R. Thurber, R. Mattis, Y. Liu, and J. J. Filliben. *The Relationship Between Resistivity and Dopant Density for Phosphorus- and Boron-Doped Silicon*. Semiconductor Measurement Technology. Washington, D.C.: U.S. Dept. of Commerce, National Bureau of standards, 1982 (cit. on p. 115).
- [109] K. T. Schütt, H. E. Sauceda, P.-J. Kindermans, A. Tkatchenko, and K.-R. Müller. "SchNet – A deep learning architecture for molecules and materials". *The Journal of Chemical Physics*, 148 (24) (2018), 241722. DOI: 10.1063/1.5019779 (cit. on p. 115).

7 REFERENCES

- [110] T. Xie and J. C. Grossman. “Crystal Graph Convolutional Neural Networks for an Accurate and Interpretable Prediction of Material Properties”. *Physical Review Letters*, 120 (14) (2018), 145301. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.120.145301 (cit. on p. 115).
- [111] C. Chen, W. Ye, Y. Zuo, C. Zheng, and S. P. Ong. “Graph Networks as a Universal Machine Learning Framework for Molecules and Crystals”. *Chemistry of Materials*, 31 (9) (2019), 3564–3572. DOI: 10.1021/acs.chemmater.9b01294 (cit. on p. 115).
- [112] J. Klicpera, S. Giri, J. T. Margraf, and S. Günnemann. *Fast and Uncertainty-Aware Directional Message Passing for Non-Equilibrium Molecules*. 2020. URL: <http://arxiv.org/pdf/2011.14115v2> (cit. on p. 115).
- [113] B. DeCost and K. Choudhary. *Atomistic Line Graph Neural Network for Improved Materials Property Predictions*. arXiv. Online, 2021. URL: <http://arxiv.org/pdf/2106.01829v1> (cit. on p. 115).
- [114] D. Jha, L. Ward, Z. Yang, C. Wolverton, I. Foster, W.-K. Liao, A. Choudhary, and A. Agrawal. “IRNet: A General Purpose Deep Residual Regression Framework for Materials Discovery”. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining – KDD ’19*. Ed. by A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis. New York, NY: ACM Press, 2019, 2385–2393. ISBN: 9781450362016. DOI: 10.1145/3292500.3330703 (cit. on p. 116).
- [115] S. Rühl. “The Inorganic Crystal Structure Database (ICSD): A Tool for Materials Sciences”. *Materials Informatics*. Ed. by O. Isayev, A. Tropsha, and S. Curtarolo. Wiley, 2019, 41–54. ISBN: 978-3-527-34121-4. DOI: 10.1002/9783527802265.ch2 (cit. on p. 116).
- [116] S. K. Kauwe, T. Welker, and T. D. Sparks. “Extracting Knowledge from DFT: Experimental Band Gap Predictions Through Ensemble Learning”. *Integrating Materials and Manufacturing Innovation*, 9 (3) (2020), 213–220. DOI: 10.1007/s40192-020-00178-0 (cit. on p. 116).

List of abbreviations

AI	artificial intelligence
CBFV	composition-based feature vector
CrabNet	Compositionally Restricted Attention-Based network
DFT	density functional theory
DL	deep learning
EDM	element-derived matrix
GPU	graphics processing unit
HPC	high performance computing
ICSD	Inorganic Crystal Structure Database
MD	molecular dynamics
MGI	Materials Genome Initiative
MI	materials informatics
ML	machine learning
PCD	Pearson's Crystal Data
TRC	Google TPU Research Cloud
XAI	explainable AI

Acknowledgements

This dissertation is based on the research performed at the Technische Universität Berlin (TU Berlin), Institute of Material Science and Technology, Fachgebiet Keramische Werkstoffe / Chair of Advanced Ceramic Materials.

First and foremost, I would like to express my profound gratitude to my thesis supervisor Professor Dr. Aleksander Gurlo of the TU Berlin for his excellent supervision, encouragement, support, and skilled guidance from the initial to the final stages of this work. My sincere thanks also go to my supervisor Mathias Czasny and my office mates Dr. Xifan Wang, Dr. Hiba Bensalah, and David Karl. I am grateful for your help, cooperation, and involvement in one way or another throughout the conception, development, refinement, and finishing of this work.

Many thanks to Dr. Maged Bekheet and Dr. Ulla Simon for their support throughout the publication and dissertation processes, and to Amanmyrat Abdullayev for the thorough review of this thesis. Special thanks also to the technicians and support staff Kai-Jens Weisser, Harald Link, Peter Schnepfmüller, Heinz Sap, Delf Kober and Anne-Claude Amtsfeld for their unending assistance and technical wizardry.

I would like to extend my deep gratitude towards all of the colleagues and fellow students at the Fachgebiet Keramische Werkstoffe for the great times in the lab, the insightful conversations, and for making this group such a great place to work at. In addition, there were many others outside of the Fachgebiet Keramische Werkstoffe who supported my work.

I express my sincere gratitude to: Dr. Taylor D. Sparks at the University of Utah, for being an excellent host during my four-month research visit in Salt Lake City, USA. Dr. Steven Kaaipuupuu Keaniani “Ka’ai” Kauwe, Dr. Jake Graser, and Ryan J. Murdock at the Sparks Group for teaching and inspiring me, involving me in your projects (both for research and for fun), and for being my lifelong friends. Joris Weigert and Dr. Pay Gießelmann for the insightful discussions and brainstorming sessions. Mahamad Salah

ACKNOWLEDGEMENTS

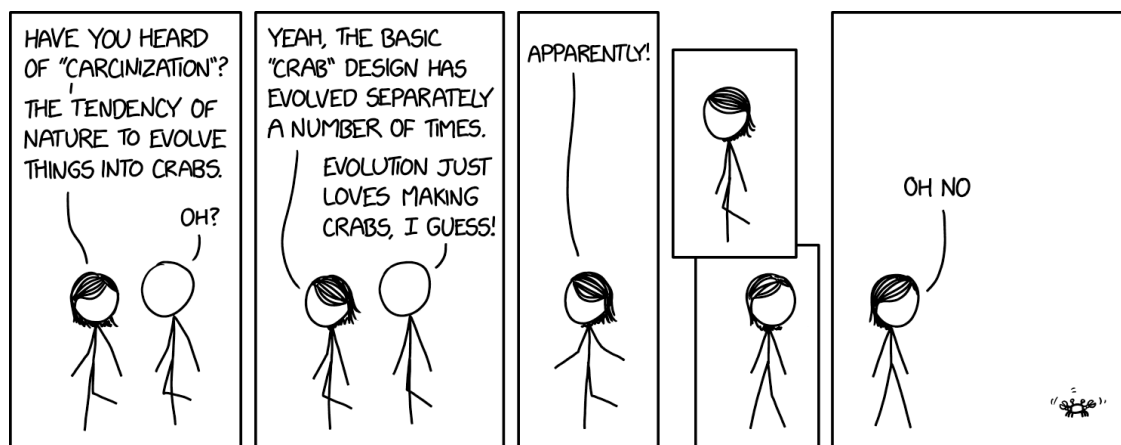
Mahmoud for being an excellent student and for supporting me in some of the research presented in this dissertation.

The large-scale models and benchmarks in this dissertation were run on the high performance computing (HPC) cluster maintained by the Institute of Mathematics (Faculty II) at the TU Berlin. Some additional research has been conducted on the HPC in the Core Facility for Advanced Research Computing at the Case Western Reserve University as well as on the Google TPU Research Cloud (TRC). Furthermore, I gratefully acknowledge the work done by the many contributors to the open-source software packages used in this work.

This research was funded by the PhD fellowship of the Berlin International Graduate School in Model and Simulation based Research (BIMoS) at the TU Berlin. The research stay in Salt Lake City, USA was partially financed by the German Academic Exchange Service (DAAD) under grant number 57438025.

I would like to thank my family and friends in Germany, Canada, Taiwan, and elsewhere around the world, for their unwavering support and encouragement. Thank you, mom, for your words of wisdom, encouragement, and never-ending support. Last but certainly not least, I express my heartfelt gratitude to my wife Nina, for always being on my side, for helping me through thick and thin, and for showing me the world.

Without any of you, this dissertation would have been impossible.



Reprinted from xkcd.com (#2314) under the CC BY-NC 2.5 license.