Lamalab Tool and Paper Notes

LamaLab

2024-04-11

Table of contents

1 Tool and paper minutes

In our group seminars, we have a tradition of dedicating a few minutes to showcase tools/software/tricks/methods that we find useful. This repository is a collection of these tool minutes.

Part I

Tools

2 Hydra

2.1 Getting started

Hydra is an open-source Python framework that simplifies the development of research and other complex applications. The key feature is the ability to dynamically create a hierarchical configuration by composition and override it through config files and the command line. The name Hydra comes from its ability to run multiple similar jobs - much like a Hydra with multiple heads.

2.1.1 Key features:

- Hierarchical configuration composable from multiple sources
- Configuration can be specified or overridden from the command line
- Dynamic command line tab completion
- Run your application locally or launch it to run remotely
- Run multiple jobs with different arguments with a single command

2.1.2 Installation

```
pip install hydra-core --upgrade
```

2.1.3 Basic example

Config, e.g., in `conf/config.yaml`:

```
db:
driver: mysql
user: omry
pass: secret
```

3 IP Rotator

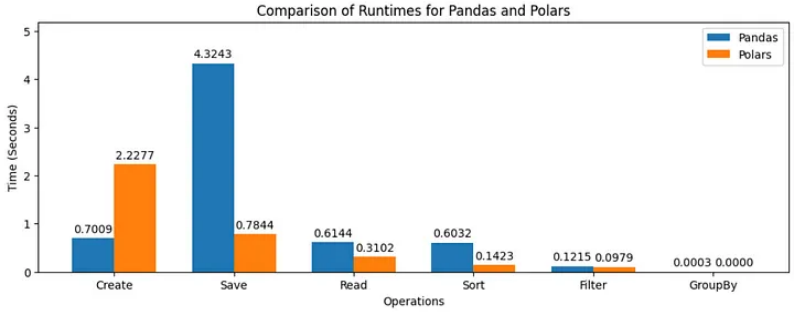
3.1 GitHub repository

iq-requests-rotator

3.1.1 Example usage:

```
import requests
from requests_ip_rotator import ApiGatewaywith ApiGateway("https://site.com") as g:
    session = requests.Session()
    session.mount("https://site.com", g)    response = session.get("https://site.com/index.php
    print(response.status_code)
```

4 Polars



4.1 An alternative to pandas

The advantages of polars can be directly seen in the image above. It is clear from the graph that Polars perform faster than Pandas for most operations. This is particularly true for the GroupBy operation, where Polars is nearly 20 times faster than Pandas. The Filter operation is also significantly faster in Polars, while Create operations are somewhat faster in Pandas. Overall, Polars seems to be a more performant library for data manipulation, particularly for large datasets.

4.2 Syntax example

```
import polars as pl

q = (
    pl.scan_csv("docs/data/iris.csv")
    .filter(pl.col("sepal_length") > 5)
    .group_by("species")
```

```
    .agg(pl.all().sum())
)

df = q.collect()
```

5 Thunder Client

Thunder Client is a lightweight alternative to Postman that can be used directly from VSCode.

You can use it to test your API endpoints.

For an example, see this video.

5.1 Installation

Install the Thunder client extension from the marketplace.

6 tmux

`tmux` is a terminal multiplexer. It lets you switch easily between several programs in one terminal, detach them (they keep running in the background) and reattach them to a different terminal. And do a lot more.

6.1 Installation

```
sudo apt install tmux
```

or on Mac

```
brew install tmux
```

6.2 Usage

Let's assume you are via ssh on a remote server and you want to run a long running process. You can use `tmux` to run the process in a session and then detach from it. You can then log out and log back in later to check on the process. Your process will still be running, even if your ssh session is closed.

6.2.1 On the remote server

```
tmux new -s myprocess
```

Then run your process. When you are done, detach from the session by pressing `Ctrl+b` and then `d`.

6.2.2 On the remote server later

```
tmux ls
```

This will list all the sessions. You can then reattach to the session you want by typing:

```
tmux attach -t myprocess
```

6.2.3 Panes

You can split your terminal into panes. This is useful if you want to run multiple processes in the same terminal. You can split the terminal vertically by pressing `Ctrl+b` and then `"` or horizontally by pressing `Ctrl+b` and then `%`.

To move panes around, you can use `Ctrl+b` and then `o` to cycle through the panes.

7 Robust statistics and Trimean

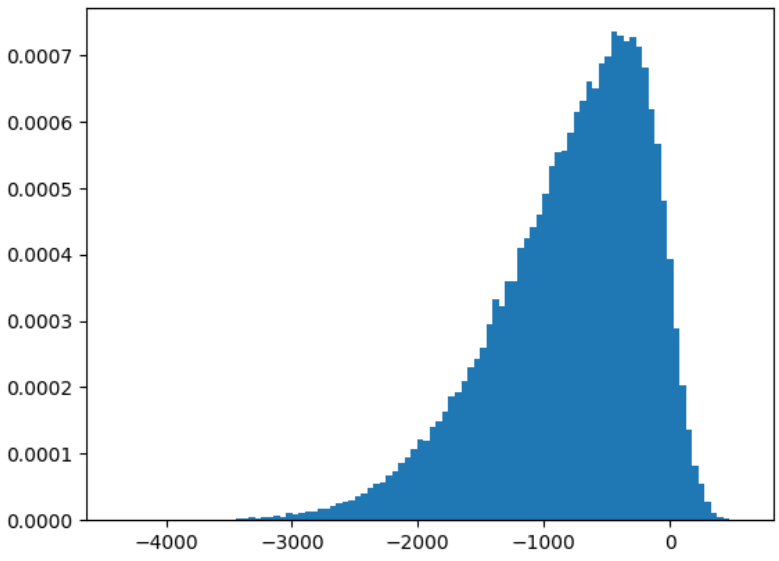
```
from scipy.stats import skewnorm
import numpy as np
import matplotlib.pyplot as plt
```

Let's generate some data that might be something we find in the real world.

```
skew_magnitude = -6
arr = skewnorm.rvs(skew_magnitude, loc=0, scale=1000, size=100000)
```

(The skew is a third-order moment.)

```
plt.hist(arr, bins=100, density=True)
plt.show()
```



Let's get a very common measure of central tendency:

```
np.mean(arr)
```

```
-789.5809069979605
```

The mean overstates the central tendency because of the skew.

The mean is defined as

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

and treats all numbers equally. No matter how big or small.

One can "fix" this by looking at "robust" statistics that are often rank based. Rank based means that we sort the data and then base our statistics on the rank of the data. In this way, they are no longer sensitive to outliers.

```
def interquartile_range(arr):
    q1 = np.percentile(arr, 25)
    q3 = np.percentile(arr, 75)
    return q3 - q1

print("Median", np.percentile(arr, 50))
print("Interquartile Range", interquartile_range(arr))
print("Mean", arr.mean())
print("Standard Deviation", arr.std())
```

```
Median -679.7024551978025
Interquartile Range 834.2816858677052
Mean -789.5809069979605
Standard Deviation 614.9363837309692
```

A very nice measure of centrality is the so-called trimean.

> "An advantage of the trimean as a measure of the center (of a distribution) is that it combines the median's emphasis on center values with the midhinge's attention to the extremes."
>
> — Herbert F. Weisberg, Central Tendency and Variability

It is defined as

$$\text{trimean} = \frac{Q_1 + 2Q_2 + Q_3}{4}$$

where Q_1 is the first quartile, Q_2 is the median, and Q_3 is the third quartile.

```
def trimean(arr):
    q1 = np.percentile(arr, 25)
    q3 = np.percentile(arr, 75)
    median = np.percentile(arr, 50)
    return (q1 + 2*median + q3)/4

print("Trimean", trimean(arr))
```

```
Trimean -708.4430042323374
```

8 Easy fast `.apply` for pandas

`apply` in `pandas` is slow. This is the case because it does not take advantage of vectorization. That means, in general, if you have something for which there is a built-in `pandas` (or `numpy`) function, you should use that instead of `apply`, because those functions will be optimized and typically vectorized.

The `pandarallel` package allows you to parallelize `apply` on a `pandas DataFrame` or `Series` object. It does this by using `multiprocessing`. However, since it uses multiple processes, it will use more memory than a simple `apply`.

If your data just barley fits in memory, you should not use `pandarallel`. However, if it does fit in memory, and you have a lot of cores, then `pandarallel` can speed up your code significantly with just changing one line of code.

```
from pandarallel import pandarallel

pandarallel.initialize(progress_bar=True)

# df.apply(func)
df.parallel_apply(func)
```

9 BFG Repo-Cleaner

If you did not take with your `.gitignore` or just used `git add .` you might have by accident committed large files. This might lead to an error like

```
remote: error: See https://gh.io/lfs for more information.
remote: error: File reports/gemini-pro/.langchain.db is 123.01 MB; this exceeds GitHub's file s
remote: error: GH001: Large files detected. You may want to try Git Large File Storage - https
To github.com:lamalab-org/chem-bench.git
 ! [remote rejected]     kjappelbaum/issue258 -> kjappelbaum/issue258 (pre-receive hook decline
error: failed to push some refs to 'github.com:lamalab-org/chem-bench.git'
```

To fix this, you need to remove the large files. A convenient tool for doing this is BFG.

Once you download the file you can run it using something like

```
java -jar ~/Downloads/bfg-1.14.0.jar --strip-blobs-bigger-than 100M --no-blob-protection
```

to remove large files.

Note that this here uses `--no-blob-protection` as BFG defaults to not touching the last commit.

After the BFG run, it will prompt you to run something like

```
git reflog expire --expire=now --all && git gc --prune=now --aggressive
```

10 showyourwork

showyourwork : https://github.com/showyourwork is a framework for building reproducible papers. The package works on a combination of Tex and Python code, where you can on the fly modify your plots.

The pre-requisites are: 1. define a conda environment with the packages are that necessary for plotting 2. use the `\script{}`, `\variable{}` and other commands to link your figures/tables to a Python script. 3. compile the paper

Part II

Papers

11 Leveraging language representation for materials exploration and discovery

11.1 Why discussing this paper?

I chose Jiaxing et al.'s paper for our journal club because

- LLMs successfully applied in other domain, interesting to see what can be done in material science
- One among the few paper where LLMs are applied in materials science for actual material discovery.
- Nice embedding figures

11.2 Context

- Material space is not completely explored. And there is possibility of finding better materials in many applications.
- ML recommender systems for exploring material spaces are already there but not many using "LLMs"
- LLM framework for recommending prototype crystal structures and later validate through first-principles calculations and experiments
- Why LLMs? - Universal task agnostic representations

11.3 Some Previous LLM Models

11.3.1 MatSciBERT

MatsciBERT was pretrained on whole sections of more than 1 million materials science articles with masked language modelling.

11.3.2 MatBERT

MatBERT was trained by sampling 50 million paragraphs from 2 million articles masked language modelling.

11.3.3 Word2Vec

Mat2Vec was trained similarly as Word2vec training through skip-gram with negative sampling. Each word is embedded into a 200-dimensional vector.

11.4 Problem setting

- Hand-crafted features and specialized structural models have limitations in providing universal and task-agnostic representations within the vast material space.
- Additional contexts are also very useful. for eg: (doping, temperature, synthesis conditions)
- In materials exploration and discovery context:

(i) effective representations of both chemical and structural complexity, (ii) successful recall of relevant candidates to property of interest

(ii) accurate candidate ranking based on multiple desired functional properties.

11.5 Approach

The authors propose a two-step funnel based approach

- 1. RECALL - Given a material finding similar material from a set of materials
- 2. RANKING - Based on functional properties rank the recalled materials

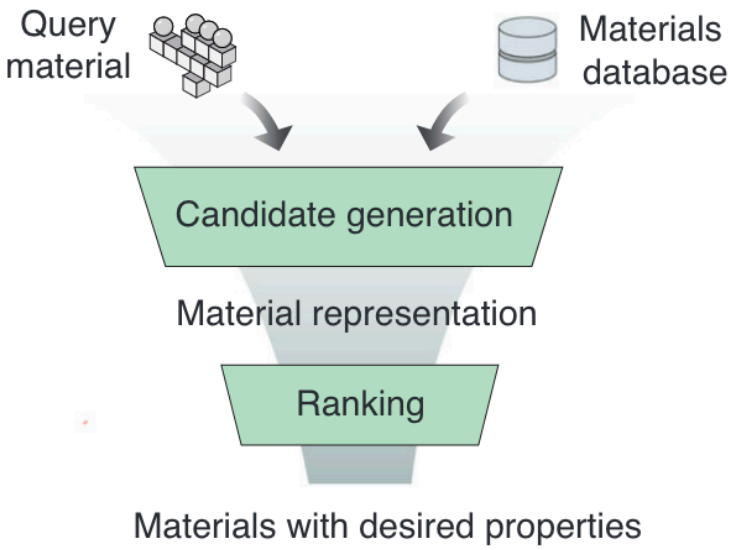


Figure 11.1: Funnel based recommender framework

11.5.1 Recalling similar materials

The authors use Robocrystallographer representation to describe the material. Encode the material description using pretrained MatBERT (compared other encoders as well), and use this as a feature vector

- Use a Query material (a well studied known material with property of interest).
- Encode all material in database and Query material (Robocrystallographer + MatBERT)

- Look at cosine similarity of feature vectors (material in database with Query material)

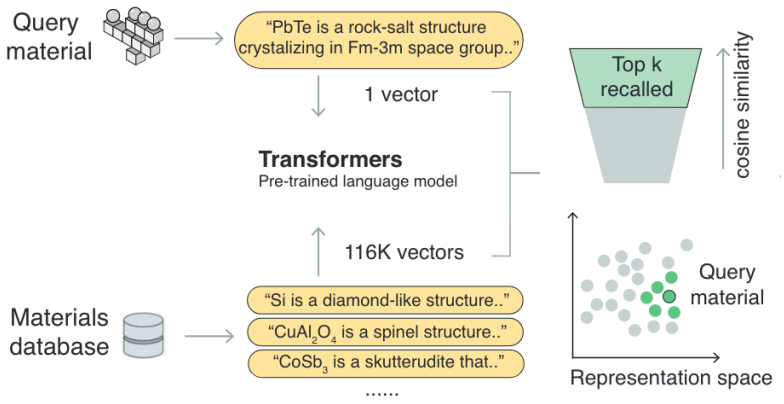


Figure 11.2: Recall material based on cosine similarity

11.5.2 Ranking potential materials

Based on multiple properties the recalled materials are ranked.

Usually for any application, and in this paper, for thermoelectric material as well many properties are important hence a ranker based on performance on different functional aspects.

- Author train a Multitask Mixture of Expert Model (MMoEM), using multitask learning to rank the materials.

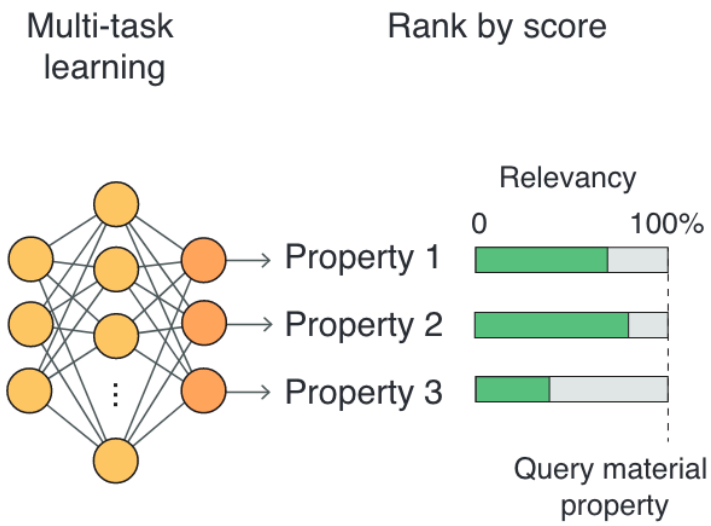


Figure 11.3: Rank material based on cosine similarity

11.6 Results

The authors perform ablations to understand the importance of the different components of their model. While there are some differences, the differences are not drastic.

11.6.1 Ablation on Representation suitable for RECALL step

Two set of models 1. Uses only composition of materials `Baseline: Mat2Vec`

A(Composition)–> B(MatBERT)

2. Uses Both composition and structure `Baseline: CrystalNN Fingerprint`

A(Material)–> B(RoboCrystallographer)–> C(MatBERT)

11.6.1.1 Embeddings from composition only and Composition + Structure

Structure level representations exhibit more distinct separation (well-defined domains) by material groups

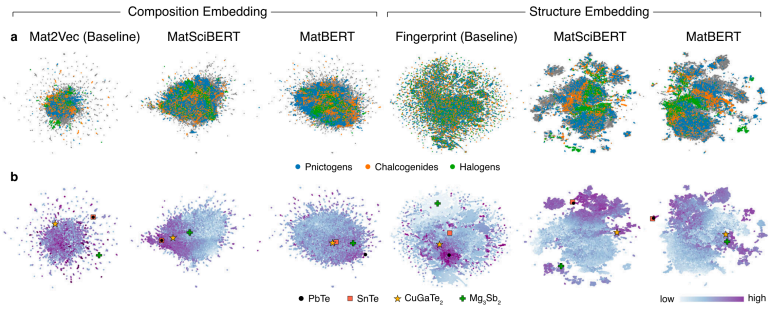


Figure 11.4: UMAP of embeddings from different representations

For further evaluation, authors evaluated material embedding performance on downstream property prediction tasks.

The task models were multi-layer perceptrons (MLPs) with meanabsolute-error (MAE) training loss.

The tasks consisted of band gap, energy per atom, bulk modulus, shear modulus, Debye temperature, and coefficient of thermal expansion from AFLOW dataset.

Composition embedding					**Structure embedding**		
Property	Metric	Mat2Vec (Baseline)	MatSciBERT	MatBERT	Fingerprint (Baseline)	MatSciBERT	MatBERT
E/atom	MAE	0.47 ± 0.02	0.42 ± 0.01	0.37 ± 0.01	1.13 ± 0.02	0.32 ± 0.02	0.29 ± 0.03
	R^2	0.81 ± 0.02	0.86 ± 0.01	0.88 ± 0.01	0.283 ± 0.02	0.95 ± 0.01	0.96 ± 0.01
E_g	MAE	0.15 ± 0.01	0.20 ± 0.02	0.19 ± 0.01	0.54 ± 0.03	0.25 ± 0.01	0.23 ± 0.01
	R^2	0.92 ± 0.02	0.88 ± 0.02	0.88 ± 0.01	0.45 ± 0.04	0.88 ± 0.01	0.89 ± 0.01
log_*K*	MAE	0.18 ± 0.01	0.18 ± 0.01	0.17 ± 0.01	0.45 ± 0.01	0.16 ± 0.01	0.15 ± 0.01
	R^2	0.83 ± 0.01	0.83 ± 0.03	0.85 ± 0.02	0.26 ± 0.02	0.90 ± 0.01	0.93 ± 0.01
log_*G*	MAE	0.20 ± 0.01	0.23 ± 0.01	0.22 ± 0.01	0.48 ± 0.01	0.24 ± 0.01	0.23 ± 0.01
	R^2	0.82 ± 0.01	0.80 ± 0.01	0.81 ± 0.02	0.29 ± 0.03	0.83 ± 0.01	0.84 ± 0.01
$\log_{10}$_θ	MAE	0.06 ± 0.01	0.07 ± 0.01	0.06 ± 0.01	0.13 ± 0.01	0.07 ± 0.01	0.06 ± 0.01
	R^2	0.81 ± 0.02	0.82 ± 0.03	0.84 ± 0.02	0.34 ± 0.05	0.85 ± 0.03	0.88 ± 0.02
$\log_{10}$_α	MAE	0.07 ± 0.01	0.07 ± 0.01	0.07 ± 0.01	0.15 ± 0.01	0.07 ± 0.01	0.06 ± 0.01
	R^2	0.78 ± 0.03	0.81 ± 0.02	0.81 ± 0.02	0.19 ± 0.02	0.87 ± 0.03	0.90 ± 0.01

E/atom Energy per atom (eV), E_g Band gap (eV), *K* Bulk modulus (GPa), *G* Shear modulus (GPa), θ Debye temperature (K), α Coefficient of thermal expansion (K^{-1}).

Figure 11.5: Property prediction using embeddings from different representations

11.6.2 Finding similar materials

Starting with known materials with favorable properties for TEs such as PbTe, we analyzed the top recalled candidates and found significantly different predicted figure-of-merit zT distributions from selected baseline representations.

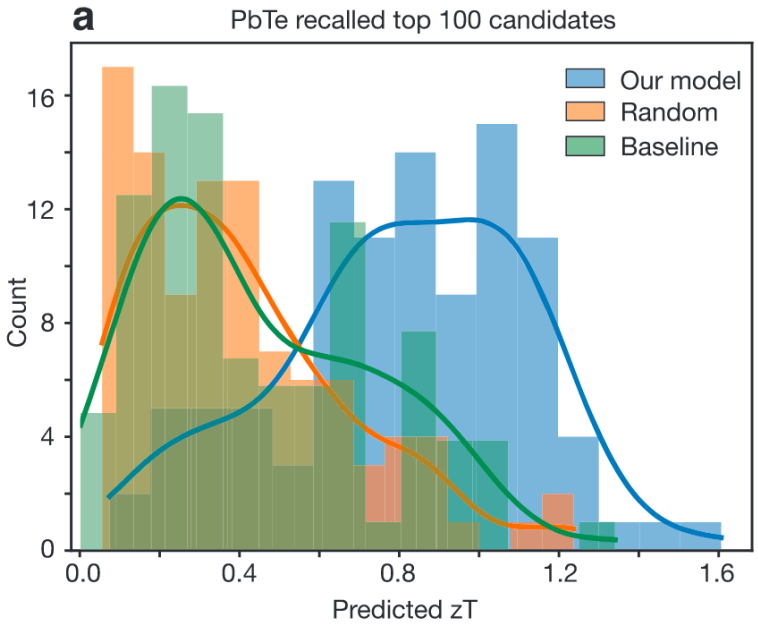


Figure 11.6: Distributions of predicted zT of the top-100 recalled candidates for PbTe as the query material predicted by MatBERT

11.6.3 Ranking potential materials

Learning from multiple related tasks provides superior performance over single-task learning by modeling task-specific objectives and cross-task relationships.

In addition to the embeddings derived from language models, the authors added further information based on context (one hot encoded temperature)

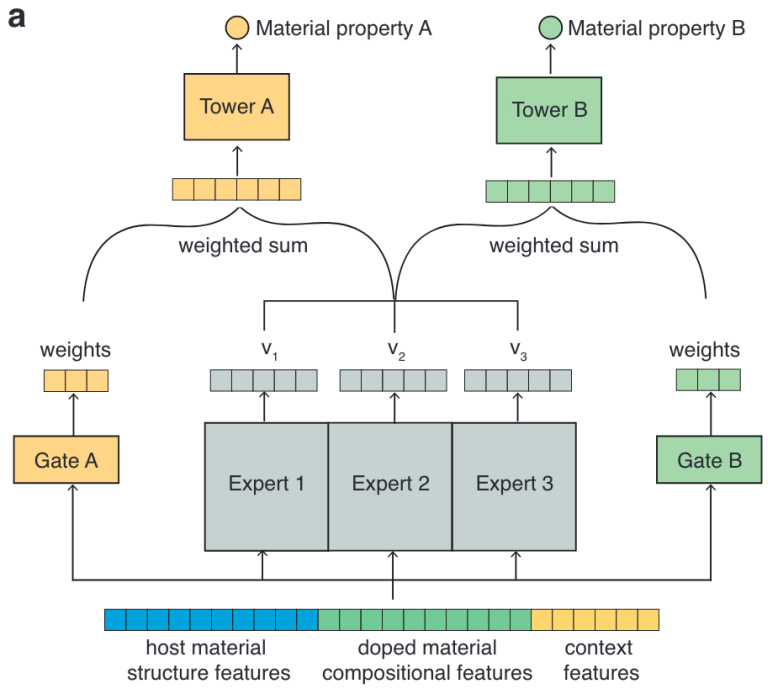


Figure 11.7: Multi-task learning framework for material property prediction.

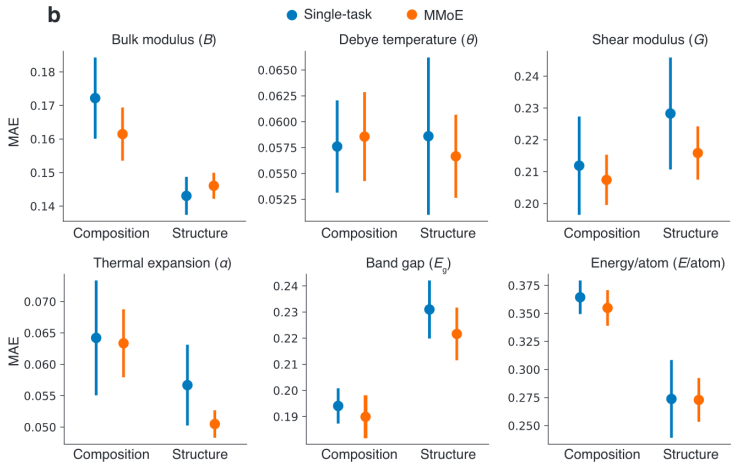


Figure 11.8: Performance for 6 material property prediction tasks between single-task models and MMoE using composition or structure embeddings.

11.7 Takeaways

- Might not need a Language model for this task
- Good to see that some of the materials where later tested in lab
- Composition vs Composition + structure not convincing.

12 Uncertainty-Aware Yield Prediction with Multimodal Molecular Features

12.1 Why discussing this paper?

I chose Chen et al.'s paper (Chen et al. 2024) for our journal club because

- An important and interesting problem in chemistry
- Uses many of the techniques we care about in our group

12.2 Context

Predicting the yield of chemical reactions is a crucial task in organic chemistry. It can help to optimize the synthesis of new molecules, reduce the number of experiments needed, and save time and resources. However, predicting the yield of a reaction is challenging due to the complexity of chemical reactions and the large number of factors that can influence the outcome.

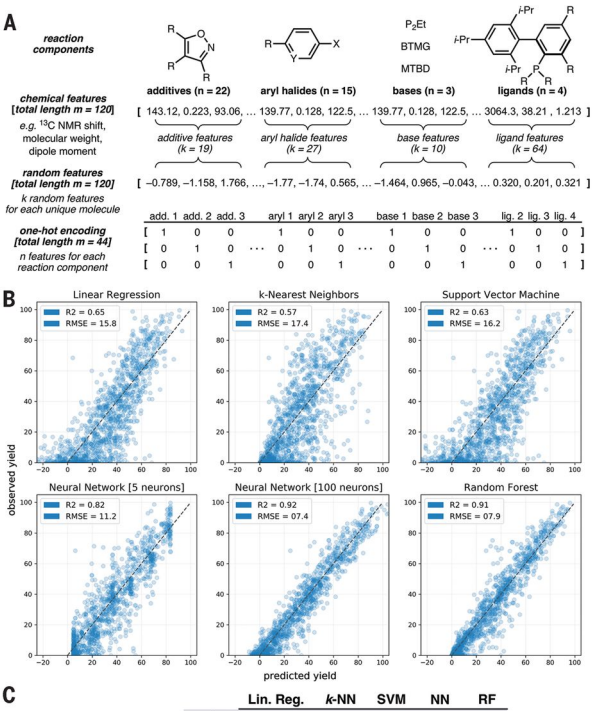
12.3 Prior work

12.3.1 Ahneman et al. (2018)

Ahneman et al. (Ahneman et al. 2018) reported in *Science* a random forest model that predicts the yield of chemical reactions in a high-throughput dataset (palladium-catalyzed Buchwald-Hartwig cross-coupling reactions). For this, the

authors created a set of features using computational techniques.

A very interesting aspect of this work is the subsequent exchange with Chuang and Keiser (Chuang and Keiser 2018) who point out that the chemical features used in the work by Ahneman et al. perform not distinguishably better than non-meaningful features.



		Lin. Reg.	*k*-NN	SVM	NN	RF
Ahneman et al.	R^2	0.66	0.64	0.64	0.93	0.92
	RMSE	15.6	16.1	16.1	7.0	7.5
Random Features	R^2	0.65	0.57	0.63	0.92	0.91
	RMSE	15.8	17.4	16.2	7.4	7.9
One-Hot Encoded	R^2	0.65	0.45	0.66	0.93	0.90
	RMSE	15.8	19.8	15.5	7.1	8.6

Figure 12.1: Figure taken from Chuang and Keiser's paper(Chuang and Keiser 2018) illustrating models trained with various featurization approaches.

12.3.2 Schwaller et al. (2020, 2021)

Schwaller et al. (Schwaller et al. 2020, 2021) utilized BERT models with a regression head to predict yields based on reac-

tion SMILES.

They observed multiple interesting effects:

- The performance on high-throughput datasets is good, on USPTO datasets the models are not predictive (R^2 on a random split of 0.117 for the gram scale)
- The yield distribution depends on the scale, which might be due to reaction at larger scale being better optimized

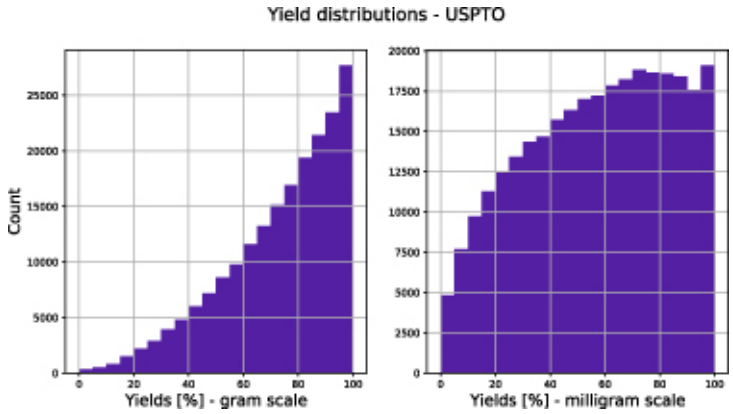


Figure 12.2: Figure taken from Schwaller et al. (Schwaller et al. 2021) illustrating the distribution of yields on different scales.

12.3.3 Kwon et al. (2022)

Kwon et al. (Kwon et al. 2022), in contrast, used graph neural networks to predict yields. They pass reactants and products through a graph neural network and concatenate the embeddings to predict the yield. They train on a similar loss as the work at hand (but use also use dropout Monte-Carlo (Gal and Ghahramani 2016) to estimate the epistemic uncertainty).

12.4 Problem setting

- prior works perform well on high-throughput datasets but not on real-world datasets

- this is partially due to a lot of noise in datasets
- of course, reaction conditions are important, too

Additionally, the authors propose that the previous representations might not be "rich" enough to capture the complexity of chemical reactions.

12.5 Approach

The authors propose to fuse multiple features. In addition, they also use a special loss function and a mixture of experts (MoE) model used to transform human-designed features.

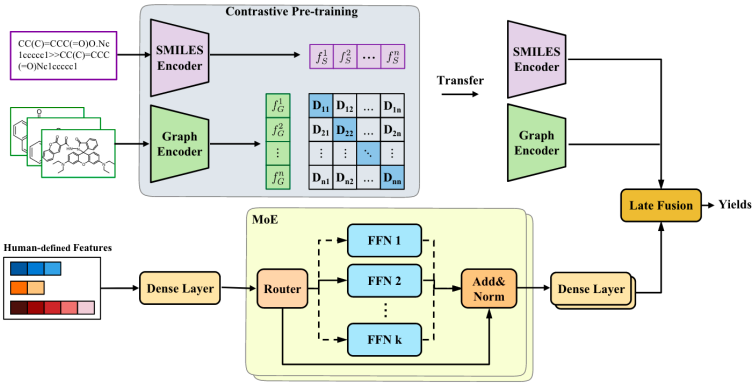


Figure 12.3: Overview of the model architecture. Figure taken from Chem et al. (Chen et al. 2024)

12.5.1 Graph encoder and SMILES encoder

The authors pretrain the graph and SMILES encoders using a contrastive loss. The graph encoder is a GNN, the SMILES encoder is a transformer.

12.5.1.1 Graph convolutional neural network

Their graph encoder is basically a message graph convolutional neural network. The authors use the `DGL` library to implement this.

The forward pass looks like this:

```
for _ in range(self.num_step_message_passing):
    node_feats = self.activation(self.gnn_layer(g, node_feats, edge_feats)).unsqueeze(0)
    node_feats, hidden_feats = self.gru(node_feats, hidden_feats)
    node_feats = node_feats.squeeze(0)
```

Where the GNN layer performs a simple operation such as

$$\mathbf{x}_i' = \Theta^\top \sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{e_{j,i}}{\sqrt{\hat{d}_j \hat{d}_i}} \mathbf{x}_j$$

where $\hat{d}_i$ is the degree of node i and Θ is a learnable weight matrix. $\mathcal{N}(i)$ is the set of neighbors of node i. $\mathbf{x}_i$ is the node embedding of node i, $e_{j,i}$ is the edge feature between node i and j.

The node embeddings are then aggregated using Set2Set pooling (Vinyals, Bengio, and Kudlur 2016).

12.5.1.2 SMILES encoder

For encoding SMILES, the use a transformer model. In their code, they seem to pass through only one transformer layer.

The forward pass looks like this:

```
x = self.token_embedding(text)
x = x + self.positional_embedding
x = x.permute(1, 0, 2)  # NLD -> LND
x = self.transformer(x)
x = x.permute(1, 0, 2)  # LND -> NLD
x = self.ln_final(x)
x = self.pooler(x[:,0,:])
```

They take the first token of the sequence and pass it through a linear layer to get the final representation.

12.5.1.3 Contrastive training

The authors use a contrastive loss to train the encoders.

$$\mathcal{L}_c = -\frac{1}{2}\log\frac{e^{\langle f_G^j, f_S^j\rangle/\tau}}{\sum_{k=1}^{N} e^{\langle f_G^j, f_S^k\rangle/\tau}} - \frac{1}{2}\log\frac{e^{\langle f_G^j, f_S^j\rangle/\tau}}{\sum_{k=1}^{N} e^{\langle f_G^k, f_S^j\rangle/\tau}},$$

In contrastive training, we try to maximize the similarity between positive pairs and minimize the similarity between negative pairs. In the equation above, f_G^j and f_S^j are the representations of the graph and SMILES of the same reaction, respectively. τ is a temperature parameter.

Such contrastive training allows to pretrain the encoders on a large dataset without labels.

> **Note**
>
> Contrastive learning is one of the most popular methods in self-supervised learning. A good overview can be found in Lilian Weng's amazing blog.

12.5.2 Human-features encoder

The authors also encode additional features with feedforward networks in a mixture of experts (MoE) model. The key idea behind MoE is that we replace "conventional layers" with "MoE layers" which are copies of the same layer. A gating network decides, based on the input, which layer to use. This is powerful if we sparsely select the experts-then only a subset of all weights are used in a given forward pass.

$$\mathrm{MoE}\left(x_H\right) = \sum_{i=1}^{t} \mathcal{G}\left(x_H\right)_i \cdot E_i\left(x_H\right)$$

This is a mixture of experts model. The authors use a gating network $\mathcal{G}$ to decide which expert to use. The experts E_i are simple feedforward networks. The gating network might be a simple softmax layer:

$$G_\sigma(x) = \text{Softmax}\left(x \cdot W_g\right)$$

in practice, one can improve that by adding sparsity (e.g. selecting top-k).

> **Note**
>
> MoE (Shazeer et al. 2017) has become popular recently as a way to scale LLMs. You might have across model names like Mixtral-8x7B (Jiang et al. 2024), which indicates that the model is a mixture of 8 experts, each of which is a 7B parameter model. The total number of parameters is 47B parameters, but the inference cost is similar to the one of a 14B parameter model. (Note however, that memory consumption is still high as all experts need to be loaded into memory.)
> This blog by Cameron Wolfe gives a good overview. You might also find Yannic Kilcher's video about Mixtral of Experts useful.

12.5.3 Fusion

The fusion of the different features is done by concatenating them

The complete forward pass looks like this:

```
r_graph_feats = torch.sum(torch.stack([self.clme.mpnn(mol) for mol in rmols]), 0)
p_graph_feats = self.clme.mpnn(pmols)
feats, a_loss = self.mlp(input_feats)
seq_feats = self.clme.transformer(smiles)
concat_feats = torch.cat([r_graph_feats, p_graph_feats, feats, seq_feats], 1)
out = self.predict(concat_feats)
```

where the `mpnn` method is the graph encoder, the `transformer` method is the SMILES encoder, and the `mlp` method is the human-features encoder.

12.5.4 Uncertainty (quantification)

The authors define the prediction as

$$\hat{y} = \mu(x) + \epsilon * \sigma(x)$$

where $\mu(x)$ is the prediction, $\sigma(x)$ is the uncertainty, and ϵ is a random variable sampled from a normal distribution.

The model is trained with a loss function that includes the uncertainty:

$$\mathcal{L}_u = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{1}{\sigma\left(x_i\right)^2} \left\| y_i - \mu\left(x_i\right) \right\|^2 + \log \sigma\left(x_i\right)^2 \right]$$

The σ term is capturing observation noise (aleatoric uncertainty).

> **Note**
>
> This loss comes from the idea of variational inference.
>
> $$\mathcal{L}(\lambda) = -\mathbb{E}_{q(\theta;\lambda)}[\log p(\mathbf{y} \mid \mathbf{x}, \theta)] + \mathrm{KL}(q(\theta;\lambda) \| p(\theta))$$
>
> In this equation, the first term is the negative log-likelihood, and the second term is the KL divergence between the approximate posterior $q(\theta;\lambda)$ and the prior $p(\theta)$. The KL divergence is a measure of how much the approximate posterior diverges from the prior. The idea is to minimize the negative log-likelihood while keeping the approximate posterior close to the prior. This is a way to quantify the uncertainty in the model.
> The idea comes from Bayesian inference, where we want to estimate the posterior distribution over the parameters of the model. In practice, this is intractable, so we use

variational inference to approximate the posterior with a simpler distribution. The posterior (which quantifies uncertainty) is typically computationally expensive to compute, so we use variational inference to approximate it with a simpler distribution, this is called variational inference. Since during training, we do some sampling, we need to perform a reparametrization trick (Kingma, Salimans, and Welling 2015) to make the gradients flow through the sampling operation.

12.6 Results

As in most ML papers, we have tables with bold numbers, e.g. for a dataset with amide coupling reactions:

Model	MAE ↓	RMSE ↓	R^2 ↑
Mordred	15.99 ± 0.14	21.08 ± 0.16	0.168 ± 0.010
YieldBert	16.52 ± 0.20	21.12 ± 0.13	0.172 ± 0.016
YieldGNN	15.27 ± 0.18	19.82 ± 0.08	0.216 ± 0.013
MPNN	16.31 ± 0.22	20.86 ± 0.27	0.188 ± 0.021
Ours	$\mathbf{14.76 \pm 0.15}$	$\mathbf{19.33 \pm 0.10}$	$\mathbf{0.262 \pm 0.009}$

Here, their model outperforms the baselines. But it is also interesting to see how well the Mordred baseline performs compared to much more complex models.

The pattern of their model being bold in tables is persistent across datasets.

12.6.1 Ablations

The authors perform ablations to understand the importance of the different components of their model. While there are some differences, the differences are not drastic (partially overlapping errorbars).

Model	MAE ↓	RMSE ↓	R^2 ↑
Ours	14.76 ± 0.15	19.33 ± 0.10	0.262 ± 0.009
w/o UQ	15.08 ± 0.13	19.63 ± 0.09	0.249 ± 0.009
w/o $\mathcal{L}_r$	14.80 ± 0.16	19.51 ± 0.10	0.261 ± 0.010
w/o MoE	15.12 ± 0.18	20.03 ± 0.13	0.230 ± 0.012
w/o Seq.	14.97 ± 0.16	19.55 ± 0.11	0.261 ± 0.010
w/o Graph	15.06 ± 0.15	19.59 ± 0.10	0.260 ± 0.009
w/o H.	15.83 ± 0.20	20.46 ± 0.18	0.212 ± 0.016

12.7 Take aways

- A lot of machinery, but not a drastic improvement
- It is the data, stupid! (It is not really clear how this is even supposed to work with information about the conditions)
- Interestingly, they didn't test USPTO or other datasets
- Their approach with frozen encoders is interesting, it would have been interesting to see learning curves to better understand the data efficiency of the approach

12.8 References

13 Structured information extraction from scientific text with large language models

13.1 Why discussing this paper?

I chose Dagdelen et al.'s paper (Dagdelen et al. 2024) for our journal club because:

- It is one of the last published papers to fine-tune a model for the data extraction task for materials science.
- It presents a very robust fine-tuning and evaluation process.
- Furthermore, they show how the current models can help with a tedious task such as it is annotating data.

13.2 Context

Extracting the unstructured scientific information from the articles that contain it can be a really arduos and time-consuming task. In the recent years, several works have shown the great potential that LLMs have to greatly accelerate this task. However, for some research fields or harder extraction schemas, the general pre-training of these models might not be enough to archieve the desired results. For such cases, fine-tuning have shown to be the adequate technique.

13.3 Prior work

13.3.1 Old ages

Several works from the Ceder group showed the complete tedious process. First, Huo et al. (Huo et al. 2019) use LDA + RF to classify text. To train the RF model they had to manually label 6000 materials paragraphs as they contain synthesis information or not.

In a following work, Kononova et al. (Kononova et al. 2019) trained a Word2Vec model, to then feed the embeddings to a BiLSTM-CRF. To train this NN they manually annotated more than 800 paragraphs word-by-word with tags about solid-state synthesis role (material, target, precursor or other). Furthermore, to classify the synthesis operations (NOT OPERATION, MIXING, HEATING, etc) they trained another NN with more annotated data. To this step they also had to **LEMMATIZED** the sentences and obtain each token's **POS**. Amazing hard work!

Similar works by Kim et al. (Kim et al. 2017, 2020; Mysore et al. 2019) in which they applied similar techniques such as word embeddings from language models, then fed to a named entity recognition model.

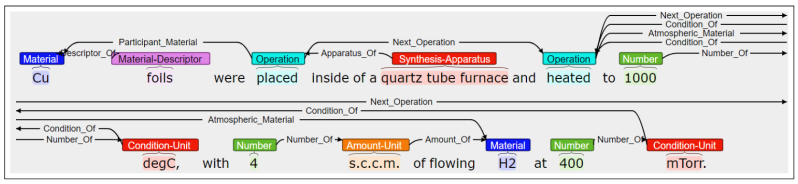


Figure 13.1: Figure taken from Mysore et al. paper (Mysore et al. 2019) illustrating how they labeled the data for the NER task.

13.3.2 ChemDataExtractor 1.0 and 2.0

Cole et al. (Swain and Cole 2016; Mavračić et al. 2021) developed ChemDataExtractor which is build from the combination of traditional ML techniques for each NLP task such as

lemmatazion, tokenization, POS tagging, it even include Table Parsing. All of these models trained in chemical text, which made this tool a really good option for extracting chemical data from text.

13.3.3 Trewartha et al. (2022)

Trewartha el al. (Trewartha et al. 2022) compared the performance of a simpler model such as a BiLSTM RNN with three more complex transformer models, BERT, SciBERT and MatBERT for the NER task. For that, they used data from three different NER datasets, each one related with different materials synthesis.

The results, showed that the more specialized BERT models were able to better recognize the different entities. However, it is important to remark that the BERT models were fine-tuned for the task.

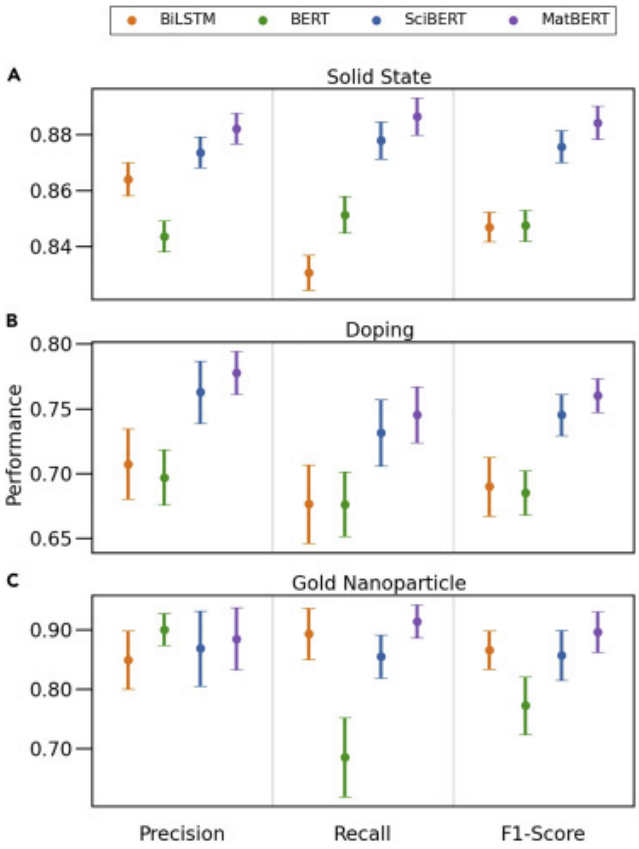


Figure 13.2: Figure taken from Trewartha et al. (Trewartha et al. 2022) summarizing the results that they obtained with each model.

13.4 Problem setting

- Almost all the scientific knowledge is contained in scientific texts in an unstructured way.
- The classical approaches include a lot of different techniques, each of them has to be trained independently.
- For those classical techniques, a lot manually labeled data is needed for each task and technique.
- LLMs appear to simplify a lot all the previous options by allowing to perform all the different NLP tasks with one unique model.

13.5 Approach

They proposed to fine-tune two models, one open-source Llama-2 70B model and a close-source one such as GPT-3, for the NER and RE tasks applied to solid-state materials. As output, they compared two different options: JSON and plain text. They proposed this for three different specificities of data: Doping, MOF and general materials data.

Task	Training samples	Completion format
Doping	413 sentences	JSON
Doping	413 sentences	English sentences
MOFs	507 abstracts	JSON
General materials	634 abstracts	JSON

13.6 Results

The results showed first of all that both models performed similar for the tasks. For the exact match, GPT-3 performed slightly better than the Llama-2 model, with overall results for both models around 50% considering all the tasks.

Task	Relation	E.M. F1 GPT-3	E.M. Llama-3
Doping	host-dopant	0.726	**0.821**
General	formula-name	**0.456**	0.367
General	formula-acronym	**0.333**	0.286
General	formula-structure/phase	**0.482**	0.470
General	formula-application	**0.537**	0.516
General	formula-description	**0.354**	0.340
MOFs	name-formula	**0.483**	0.276
MOFs	name-guest specie	**0.616**	0.408
MOFs	name-application	**0.573**	0.531
MOFs	name-description	**0.404**	0.389

It is important to comment that the exact match is an approximate lower bound on information extraction performance, since it not consider some cases such as “Lithium ion” named

(2) The results presented in this table include for both NER and RE NLP tasks.

as “Li-ion”, or MOF names such as “ZIF-8” that are described as “mesostructured MOFs formed by Cu2+ and 5hydroxy-1,3-benzenedicarboxylic acid”.

For correctly measure those ambiguities, they did a manual evaluation on a randomly sampled 10% of the test set. These results showed that the score for the extraction was much better than the showed by the exact match. This also showed that some kind of normallization proccess is needed to correctly evaluate this type of extraction tasks.

For the Doping task, three different output schema were consider, *DopingEnglish*, *DopingJSON* and *DopingExtra-English*. They compared the results for the three schema GPT-3 and Llama-2 fine-tuned models with other older models such as MatBERT and Seq2rel.

The results showed that the Llama-2 model return the best results for this task, which are slightly better than the GPT-3 ones. Both LLMs improved by far the other two models.

The difference between *DopingEnglish* and *DopingExtra-English* is that the last one include some additional information and not only the host-entity relation.

Model	Schema	E.M. Precision	E.M. Recall	E.M. F1
MatBERT	n/a	0.377	0.403	0.390
Seq2rel	n/a	0.420	0.605	0.496
GPT-3	Doping-JSON	0.772	0.684	0.725
GPT-3	Doping-English	0.803	0.754	0.778
GPT-3	DopingExtra-English	0.820	0.798	0.809
Llama-2	Doping-JSON	**0.836**	0.807	**0.821**
Llama-2	Doping-English	0.787	**0.842**	0.814
Llama-2	DopingExtra-English	0.694	0.815	0.750

A limitation of the method could be that for each of the three extractions tasks, they have to annotate between 100 and 500

text passages. This can be a tedious work. However, to overcome this limitation, they proposed to include human-in-the-loop annotation.

13.6.1 Human-in-the-loop

To overcome the limitation of having to manually annotate all the data needed for the fine-tuning process, they sucesfully implemented human-in-the-loop annotation. For that, they fine-tune the model with a small amount of manually labelled data. Then the model is asked to extract data from the other text passages. The returned data by the model is corrected by an human annotator and is feed into the model to further fine-tune it.

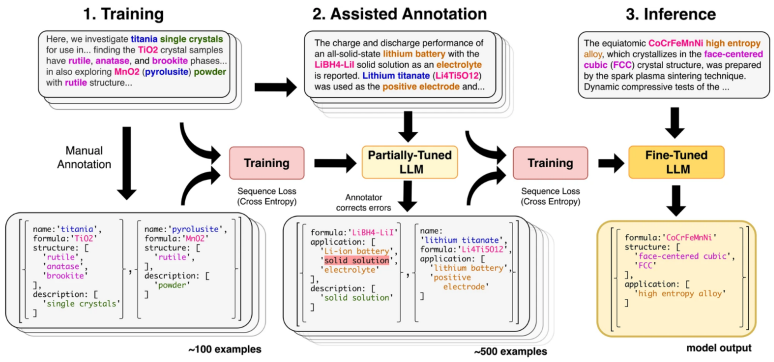


Figure 13.3: Figure showing the process used to implement human-in-the-loop annotation.

By using this technique, they greatly reduce the amount of time needed to annotate the last pieces of text compared with the first ones.

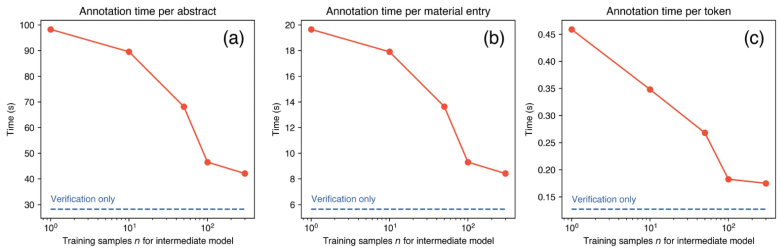


Figure 13.4: Figure showing the time reduction across the process of annotation using the human-in-the-loop technique.

By using this annotation method they greatly improve the annotation time solving one of the main drawbacks of fine-tuning an LLM. This great limitation can be seen in another works such as the one by Guo el al. (Guo et al. 2021) in which they employed 13 graduate and postdoc students to annotate about chemical reactions. After that, they have to even check all the annotation. They estimate that this process took them almost 300 hours.

13.7 Take aways

- Open source models with proper tuning can yield high-quality results similar to those of closed source models.
- Despite some labeled data is needed, the process is simplified a lot with the use of LLMs.
- With the fasst and continuous development of the current models, maybe fine-tuning for a simpler task such as data extraction is no furhter needed.

13.8 References

14 Molecular contrastive learning of representations (MolCLR) via graph neural networks

14.1 Why discussing this paper?

I chose Wang's et al.'s paper (**wang2022molecular?**) for our journal club because

- Supervised learning relies heavily on labeled training data, which can be expensive and time-consuming to obtain.
- Labeled datasets are limited; training models on such datasets might lead to overfitting and poor generalization.
- Self-Supervised Learning (SSL) provides a promising alternative. It enables learning from unlabeled data, which is much easier to acquire in real-world applications and is part of a large research effort.
- SSL learns the inherent structure and patterns in unlabeled data. By doing so, self-supervised models can acquire rich representations and knowledge that can be transferred to downstream tasks, even with limited labeled data.
- Contrastive self-supervised learning, as the name implies, is a self-supervised method that learns representations by contrasting positive and negative pairs.

14.2 Context

The paper introduces MolCLR, a self-supervised learning framework that uses graph encoders to learn differentiable molecular representations.

The authors used a large unlabeled dataset with 10 million unique molecule SMILES from ChemBERTa and PubChem. This framework involves two important steps. First, in the pretraining phase, they build molecule graphs and develop graph neural network (GNN) encoders to learn differentiable representations. Second, the pretrained GNN backbone is used for supervised learning tasks.

14.3 Main idea behind contrastive learning

The aim of contrastive representation learning is to develop an embedding space where similar samples are positioned closely together, whereas dissimilar samples are kept distant from each other.

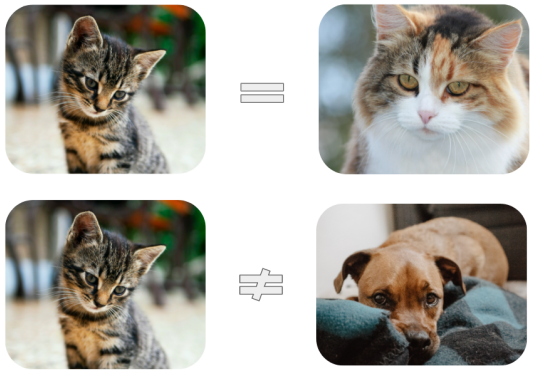


Figure 14.1: Figure taken from the blog post by Ekin Tiu illustrating the idea behind contrastive learning.

14.4 NT-Xent loss

The authors use a NT-Xent loss for the contrastive learning.

$$L_{ij} = -\log\left(\frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbf{1}_{\{k\neq i\}} \exp(\text{sim}(z_i, z_k)/\tau)}\right)$$

- `z_i` and `z_j`: Latent vectors extracted from a positive data pair.
- `N`: Batch size, indicating the number of data pairs used.
- `sim( )`: Function measuring the similarity between two vectors.
- : The temperature parameter, used to scale the similarity measures in the function.

To get more understanding, (**le2020contrastive?**) is a review paper on contrastive learning.

14.5 Overview of MolCLR framework

- The SMILES are converted into graphs in a batch. The graphs consist of nodes and edges. Nodes represent atoms, and edges represent the chemical bonds of a molecule.
- The authors studied three types of augmentations: atom masking, bond deletion, and sub-graph removal. These augmentations introduce variance to the model, allowing it to learn different substructures/topologies of the same molecule and generalize well to unseen molecules.
- The GNNs operate based on a message-passing framework. At each node, the local neighborhood information is aggregated and updated iteratively, resulting in node embeddings of the molecule.
- These embeddings are fed into a Multi-Layer Perceptron (MLP) with hidden units to obtain a latent representation of the molecule. The MLP serves as a projection head, mapping the high-dimensional representations from the encoder (GNN) to a lower-dimensional latent space.

This projection helps in learning more compact and discriminative representations for the contrastive loss.

- The key difference between supervised GNNs and self-supervised contrastive GNNs is the training objective. In a supervised setup, where we have access to ground truth labels, we can train the network to optimize standard loss functions such as Binary Cross Entropy (BCE) for classification or Mean Absolute Error/Mean Squared Error (MAE/MSE) for regression. In self-supervised contrastive learning, we aim to learn a latent space where positive pairs are closer and negative pairs are further apart.
- The goal is to score the agreement between positive pairs higher than that of negative pairs. For a given graph, its positive pair is constructed using data augmentations, while all other graphs in the batch constitute negative pairs.

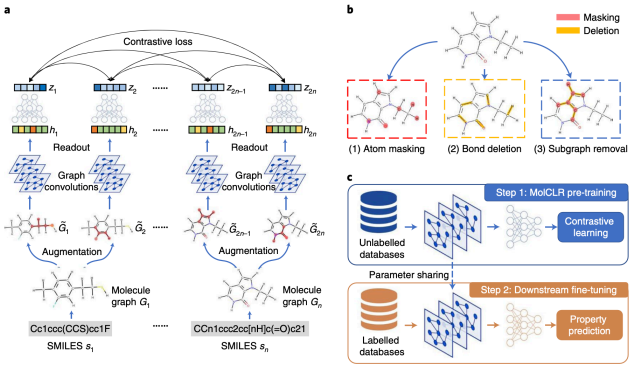


Fig. 1 | Overview of MolCLR. a, MolCLR pre-training. A SMILES s_n from a batch of N molecule data is converted to a molecule graph G_n. Two stochastic molecule graph data augmentation operators are applied to each graph, resulting two correlated masked graphs: $\tilde{G}_{2n-1}$ and $\tilde{G}_{2n}$. A base feature encoder built on graph convolutions and the readout operation extracts the representation h_{2n-1}, h_{2n}. Contrastive loss is utilized to maximize agreement between the latent vectors z_{2n-1}, z_{2n} from the MLP projection head. **b**, Molecule graph augmentation strategies: atom masking, bond deletion and subgraph removal. **c**, The whole MolCLR framework. GNNs are first pre-trained via MolCLR to learn representative features. Fine-tuning for downstream molecular property predictions shares the pre-trained parameters of the GNN encoder and randomly initializes an MLP head. It then follows the supervised learning to train the model.

Figure 14.2: Overview figure for the MolCLR framework

For more information about GNNs and their applications, check out this review paper [zhou2020graph].

14.6 Results

The authors tested the performance of MolCLR framework on seven benchmarks for classification tasks and six benchmarks for regression tasks.

The authors claim that on classification tasks with other self-supervised learning or pre-training strategies, their MolCLR framework achieves the best performance on five out of seven benchmarks, with an average improvement of 4.0 percent in (ROC-AUC (%)).

Table 1 | Test performance of different models on seven classification benchmarks

Dataset	BBBP	Tox21	ClinTox	HIV	BACE	SIDER	MUV
Molecules	**2,039**	**7,831**	**1,478**	**41,127**	**1,513**	**1,427**	**93,087**
Tasks	**1**	**12**	**2**	**1**	**1**	**27**	**17**
RF	71.4±0.0	76.9±1.5	71.3±5.6	78.1±0.6	**86.7±0.8**	**68.4±0.9**	63.2±2.3
SVM	72.9±0.0	**81.8±1.0**	66.9±9.2	**79.2±0.0**	86.2±0.0	68.2±1.3	67.3±1.3
GCN[17]	71.8±0.9	70.9±2.6	62.5±2.8	74.0±3.0	71.6±2.0	53.6±3.2	71.6±4.0
GIN[18]	65.8±4.5	74.0±0.8	58.0±4.4	75.3±1.9	70.1±5.4	57.3±1.6	71.8±2.5
SchNet[19]	84.8±2.2	77.2±2.3	71.5±3.7	70.2±3.4	76.6±1.1	53.9±3.7	71.3±3.0
MGCN[53]	**85.0±6.4**	70.7±1.6	63.4±4.2	73.8±1.6	73.4±3.0	55.2±1.8	70.2±3.4
D-MPNN[20]	71.2±3.8	68.9±1.3	**90.5±5.3**	75.0±2.1	85.3±5.3	63.2±2.3	**76.2±2.8**
Hu et al.[45]	70.8±1.5	78.7±0.4	78.9±2.4	80.2±0.9	85.9±0.8	65.2±0.9	81.4±2.0
N-Gram[44]	**91.2±3.0**	76.9±2.7	85.5±3.7	**83.0±1.3**	87.6±3.5	63.2±0.5	81.6±1.9
$MolCLR_{GCN}$	73.8±0.2	74.7±0.8	86.7±1.0	77.8±0.5	78.8±0.5	66.9±1.2	84.0±1.8
$MolCLR_{GIN}$	73.6±0.5	**79.8±0.7**	**93.2±1.7**	80.6±1.1	**89.0±0.3**	**68.0±1.1**	**88.6±2.2**

The first seven models are supervised learning methods and the last four are self-supervised/pre-training methods. Mean and standard deviation of test ROC-AUC (%) on each benchmark are reported. Best performing supervised and self-supervised/pre-training methods for each benchmark are marked as bold.

Figure 14.3: Table 1 shows the test performance on seven benchmarks on classifiaction task

The authors also claim that on regression tasks, MolCLR surpasses other pre-training baselines in five out of six benchmarks and achieves almost the same performance on the remaining ESOL benchmark. For the QM9 dataset, MolCLR does not have comparable performancee with SchNet and MGCN supervised models.

Table 2 | Test performance of different models on six regression benchmarks

Dataset	FreeSolv	ESOL	Lipo	QM7	QM8	QM9
Molecules	**642**	**1,128**	**4,200**	**6,830**	**21,786**	**130,829**
Tasks	**1**	**1**	**1**	**1**	**12**	**8**
SVM	3.14 ± 0.00	1.50 ± 0.00	0.82 ± 0.00	156.9 ± 0.0	0.0543 ± 0.0010	24.613 ± 0.144
GCN[17]	2.87 ± 0.14	1.43 ± 0.05	0.85 ± 0.08	122.9 ± 2.2	0.0366 ± 0.0011	5.796 ± 1.969
GIN[18]	2.76 ± 0.18	1.45 ± 0.02	0.85 ± 0.07	124.8 ± 0.7	0.0371 ± 0.0009	4.741 ± 0.912
SchNet[19]	3.22 ± 0.76	1.05 ± 0.06	0.91 ± 0.10	**74.2 ± 6.0**	0.0204 ± 0.0021	0.081 ± 0.001
MGCN[53]	3.35 ± 0.01	1.27 ± 0.15	1.11 ± 0.04	77.6 ± 4.7	0.0223 ± 0.0021	**0.050 ± 0.002**
D-MPNN[20]	2.18 ± 0.91	**0.98 ± 0.26**	**0.65 ± 0.05**	105.8 ± 13.2	**0.0143 ± 0.0022**	3.241 ± 0.119
Hu et al.[45]	2.83 ± 0.12	1.22 ± 0.02	0.74 ± 0.00	110.2 ± 6.4	0.0191 ± 0.0003	4.349 ± 0.061
N-Gram[44]	2.51 ± 0.19	**1.10 ± 0.03**	0.88 ± 0.12	125.6 ± 1.5	0.0320 ± 0.0032	7.636 ± 0.027
$MolCLR_{GCN}$	2.39 ± 0.14	1.16 ± 0.00	0.78 ± 0.01	**83.1 ± 4.0**	0.0181 ± 0.0002	3.552 ± 0.041
$MolCLR_{GIN}$	**2.20 ± 0.20**	**1.11 ± 0.01**	**0.65 ± 0.08**	87.2 ± 2.0	**0.0174 ± 0.0013**	**2.357 ± 0.118**

The first seven models are supervised learning methods and the last four are self-supervised/pre-training methods. Mean and standard deviation of test RMSE (for FreeSolv, ESOL, Lipo) or MAE (for QM7, QM8 and QM9) are reported. Best performing supervised and self-supervised/pre-training methods for each benchmark are marked as bold.

Figure 14.4: Table 2 shows the test performance on seven benchmarks on classifiaction task

14.7 Analysis of molecule graph augmentations

The authors employed four augmentation strategies on the classification benchmarks. Out of four augmentations, sub-graph removal with probability 0.25 achieved good (ROC-AUC %) on all of the classification benchmarks except in the BBBP benchmark as shown in the figure 2. The reason might be that model structures are sensitive in BBBP benchmark.

They also tested GIN supervised models with and without molecular graph augmentations. With augmentations (sub-graph removal with a probability of 0.25), the model achieved superior performance compared to without augmentations.

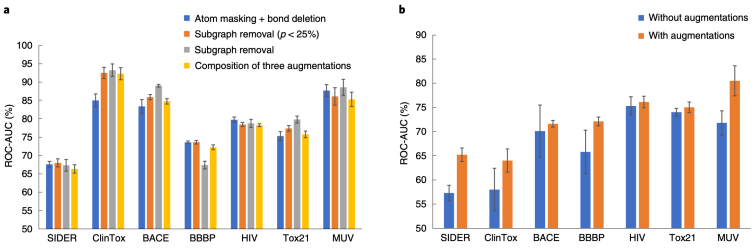


Fig. 2 | Investigation of molecule graph augmentations on classification benchmarks. a, Test performance of MolCLR models with different compositions of molecule graph augmentation strategies. **b**, Test performance of GIN models trained via supervised learning with and without molecular graph augmentations. The height of each bar represents the mean ROC-AUC (%) on the benchmark, and the length of each error bar represents the standard deviation.

Figure 14.5: Figure 2 shows the investigation of molecule graph augmentations on classification benchmarks

14.8 Investigation of MOlCLR representation

The authors examined the representations learned by pre-trained MOlCLR using t-SNE embedding. This method groups similar chemicals together in two-dimensional space. In Figure 3, they show a picture of 100,000 molecules from the validation set of PubChem database showing similar/dissimilar molecules learned by MolCLR pretraining.

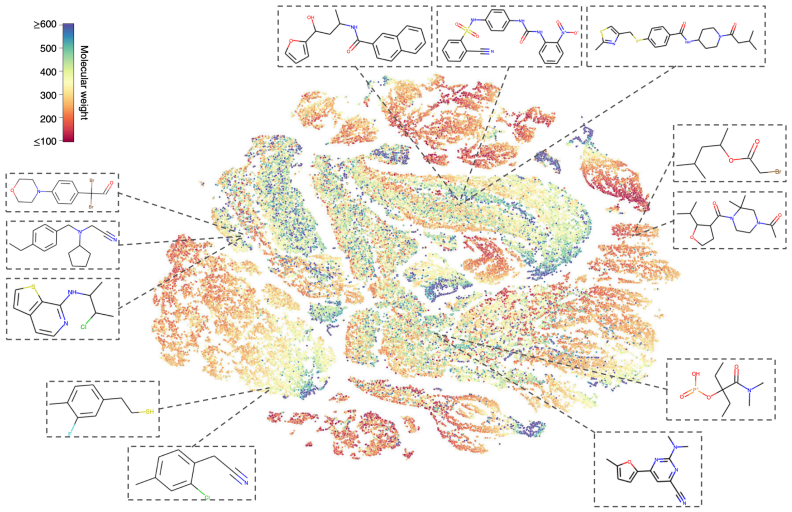


Fig. 3 | Visualization of molecular representations learned by MolCLR via t-SNE. Representations are extracted from the validation set of the pre-training dataset, which contains 100,000 unique molecules. Each point is coloured by its corresponding molecular weight (g mol^{-1}). Some molecules close in the representation domain are also shown.

Figure 14.6: Figure 3 Visualization of molecular representations learned by MolClr via t-sne

For instance, the three molecules on the top possess carbonyl groups connected with aryls. The two molecules on the bottom left have similar structures, where a halogen atom (fluorine or chlorine) is connected to benzene.

Therefore, learned representations are not random but they are meaningful.

In addition to this, the authors also query molecule from the PubcChem with ID 42953211 and the closest nine similar molecules are retrevied along with RDKFP and ECFP similarities labelled. It is observed that these selected molecules share the same functional groups, including alkyl halides (chlorine), tertiary amines, ketones and aromatics. A thiophene structure can also be found in all the molecules.

More examples of query molecules can be found in supplementary information of this paper.

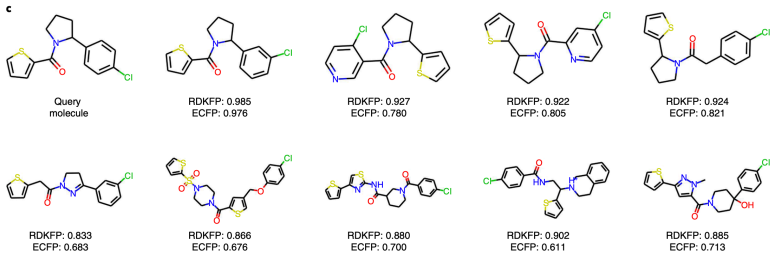


Figure 14.7: Figure shows Comparison of MolCLR-learned representations and conventional FPs using the query molecule (PubChem ID 42953211)

14.9 Take aways

- Good molecular representations are important for better predictions
- Self-supervised contrastive learning would be an advantage for generalizable machine learning over supervised learning

References

Ahneman, Derek T., Jesús G. Estrada, Shishi Lin, Spencer D. Dreher, and Abigail G. Doyle. 2018. "Predicting Reaction Performance in c–n Cross-Coupling Using Machine Learning." *Science* 360 (6385): 186–90. https://doi.org/10.1126/science.aar5169.

Chen, Jiayuan, Kehan Guo, Zhen Liu, Olexandr Isayev, and Xiangliang Zhang. 2024. "Uncertainty-Aware Yield Prediction with Multimodal Molecular Features." *Proceedings of the AAAI Conference on Artificial Intelligence* 38 (8): 8274–82. https://doi.org/10.1609/aaai.v38i8.28668.

Chuang, Kangway V., and Michael J. Keiser. 2018. "Comment on 'Predicting Reaction Performance in c–n Cross-Coupling Using Machine Learning'." *Science* 362 (6416). https://doi.org/10.1126/science.aat8603.

Dagdelen, John, Alexander Dunn, Sanghoon Lee, Nicholas Walker, Andrew S. Rosen, Gerbrand Ceder, Kristin A. Persson, and Anubhav Jain. 2024. "Structured Information Extraction from Scientific Text with Large Language Models." *Nature Communications* 15 (1): 1418. https://doi.org/10.1038/s41467-024-45563-x.

Gal, Yarin, and Zoubin Ghahramani. 2016. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning." In *International Conference on Machine Learning*, 1050–59. PMLR.

Guo, Jiang, A. Santiago Ibanez-Lopez, Hanyu Gao, Victor Quach, Connor W. Coley, Klavs F. Jensen, and Regina Barzilay. 2021. "Automated Chemical Reaction Extraction from Scientific Literature." *Journal of Chemical Information and Modeling* 62 (9): 2035–45. https://doi.org/10.1021/acs.jcim.1c00284.

Huo, Haoyan, Ziqin Rong, Olga Kononova, Wenhao Sun, Tiago Botari, Tanjin He, Vahe Tshitoyan, and Gerbrand Ceder. 2019. "Semi-Supervised Machine-Learning Classification of Materials Synthesis Procedures." *Npj Computational Materials* 5 (1). https://doi.org/10.1038/s41524-019-0204-1.

Jiang, Albert Q., Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, et al. 2024. "Mixtral of Experts." https://arxiv.org/abs/2401.04088.

Kim, Edward, Kevin Huang, Adam Saunders, Andrew McCallum, Gerbrand Ceder, and Elsa Olivetti. 2017. "Materials Synthesis Insights from Scientific Literature via Text Extraction and Machine Learning." *Chemistry of Materials* 29 (21): 9436–44. https://doi.org/10.1021/acs.chemmater.7b03500.

Kim, Edward, Zach Jensen, Alexander van Grootel, Kevin Huang, Matthew Staib, Sheshera Mysore, Haw-Shiuan Chang, et al. 2020. "Inorganic Materials Synthesis Planning with Literature-Trained Neural Networks." *Journal of Chemical Information and Modeling* 60 (3): 1194–1201. https://doi.org/10.1021/acs.jcim.9b00995.

Kingma, Diederik P., Tim Salimans, and Max Welling. 2015. "Variational Dropout and the Local Reparameterization Trick." https://arxiv.org/abs/1506.02557.

Kononova, Olga, Haoyan Huo, Tanjin He, Ziqin Rong, Tiago

Botari, Wenhao Sun, Vahe Tshitoyan, and Gerbrand Ceder. 2019. "Text-Mined Dataset of Inorganic Materials Synthesis Recipes." *Scientific Data* 6 (1). https://doi.org/10.1038/s41597-019-0224-1.

Kwon, Youngchun, Dongseon Lee, Youn-Suk Choi, and Seokho Kang. 2022. "Uncertainty-Aware Prediction of Chemical Reaction Yields with Graph Neural Networks." *Journal of Cheminformatics* 14 (1). https://doi.org/10.1186/s13321-021-00579-z.

Mavračić, Juraj, Callum J. Court, Taketomo Isazawa, Stephen R. Elliott, and Jacqueline M. Cole. 2021. "ChemDataExtractor 2.0: Autopopulated Ontologies for Materials Science." *Journal of Chemical Information and Modeling* 61 (9): 4280–89. https://doi.org/10.1021/acs.jcim.1c00446.

Mysore, Sheshera, Zach Jensen, Edward Kim, Kevin Huang, Haw-Shiuan Chang, Emma Strubell, Jeffrey Flanigan, Andrew McCallum, and Elsa Olivetti. 2019. "The Materials Science Procedural Text Corpus: Annotating Materials Synthesis Procedures with Shallow Semantic Structures." https://arxiv.org/abs/1905.06939.

Schwaller, Philippe, Alain C Vaucher, Teodoro Laino, and Jean-Louis Reymond. 2020. "Data Augmentation Strategies to Improve Reaction Yield Predictions and Estimate Uncertainty." *Chemrxiv Preprint*.

———. 2021. "Prediction of Chemical Reaction Yields Using Deep Learning." *Machine Learning: Science and Technology* 2 (1): 015016.

Shazeer, Noam, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer." https://arxiv.org/abs/1701.06538.

Swain, Matthew C., and Jacqueline M. Cole. 2016. "ChemDataExtractor: A Toolkit for Automated Extraction of Chemical Information from the Scientific Literature." *Journal of Chemical Information and Modeling* 56 (10): 1894–904. https://doi.org/10.1021/acs.jcim.6b00207.

Trewartha, Amalie, Nicholas Walker, Haoyan Huo, Sanghoon Lee, Kevin Cruse, John Dagdelen, Alexander Dunn, Kristin A. Persson, Gerbrand Ceder, and Anubhav Jain. 2022. "Quantifying the Advantage of Domain-

Specific Pre-Training on Named Entity Recognition Tasks in Materials Science." *Patterns* 3 (4): 100488. https://doi.org/10.1016/j.patter.2022.100488.

Vinyals, Oriol, Samy Bengio, and Manjunath Kudlur. 2016. "Order Matters: Sequence to Sequence for Sets." https://arxiv.org/abs/1511.06391.