Do large language models know chemistry?

Andrew D. White,[1, *] Glen M. Hocky,[2] Heta A. Gandhi,[1] Mehrad Ansari,[1] Sam Cox,[1] Geemi P. Wellawatte,[3] Subarna Sasmal,[2] Ziyue Yang,[1] Kangxin Liu,[2] Yuvraj Singh,[2] and Willmor J. Peña Ccoa[2]

[1]*Department of Chemical Engineering, University of Rochester*
[2]*Department of Chemistry, New York University[†]*
[3]*Department of Chemistry, University of Rochester*
(Dated: July 6, 2022)

Mostly yes. We systematically evaluate machine learning large language models (LLMs) that generate code in the context of chemistry. We produce a benchmark set of problems, and evaluate these models based on correctness of code by automated testing and evaluation by experts. We find recent LLMs are able to write correct code across a variety of topics in chemistry and their accuracy can be increased by 30 percentage points via prompt engineering strategies, like putting copyright notices at the top of files. These dataset and evaluation tools are open source which can be contributed to or built upon by future researchers, and will serve as a community resource for evaluating the performance of new models as they emerge. We also describe some good practices for employing LLMs in chemistry. The general success of these models demonstrates that their impact on chemistry teaching and research is poised to be enormous.

I. INTRODUCTION

Large language models (LLMs) are multi-billion parameter transformer neural networks[1] that are trained on enormous collections of documents (corpus) without supervision or labels.[2] LLMs can do multiple tasks like classifying natural language, translating text, and document search. Perhaps the most remarkable task of LLMs is to complete an input string of text; via this mechanism (called causal language modeling), LLMs can write unit tests, document function, write code from a doc string, answer questions, and complete stoichiometric equations.[3,4]

We previously discussed the outlook of LLMs in chemistry.[5] In the few months since then, LLMs have been both developed for specific chemistry problems[6,7] and general LLMs have been applied in chemistry.[8,9] An open question for LLMs such as GPT-3,[3] T5,[10] or GPT-neo[11] that are trained on very large and varied textual data is if they can be applied in domains like chemistry, which have specialized language and knowledge. In our initial work,[5] we found relationships between SMILES and natural language was possible with GPT-3. SMILES is the the standard method of representing molecules as strings.[12] It is even possible to loosely edit structures via natural language (see Fig. 5).[13,14] However, the extent to which LLMs can be directly applied in chemistry in the broad context of research and teaching is unexplored. The large amount of specific domain knowledge required to solve chemistry problems may limit applicability of general LLMs. For example, recent work has found that knowledge of the periodic table of elements requires very high parameter counts.[4]

Recent comparisons of LLMs that generate code can

be found Ref 15. Here, we focus our study on whether LLMs that generate code[16] can be applied to chemistry tasks of a computational nature (both computational chemistry problems, and general tasks which can be expressed as simple computer programs, such as ranking elements by ionic radius). Most LLMs that generate computer code are causal decoder-only models[16–18]—a user provides a sequence of text (called the prompt) and it proposes a continuation of the text (the completion).[19] There are LLMs trained on code that can infill or match encoder/decoder natural language to code like CodeBERT,[20] but they are typically used for embedding code for tasks like classification, document retrieval, or translating code to natural language.

Evaluating LLMs' knowledge of chemistry should be distinguished from capability to reason or understand. LLMs can make compelling completions, but are incapable of reasoning and demonstrate superficial understanding.[21,22] Our goal is to evaluate LLMs' ability to correlate natural language, equations, code, and heuristics of chemistry.

II. METHODS

We have compiled a categorized set of chemistry and related example prompts for benchmarking code-generating LLMs in a public repository.[23] The examples span a range of topics that we consider common programming tasks by students or researchers, with an emphasis on the discipline of computational chemistry. The distribution of topics is given in Tab. I. Some examples were labeled as only expert evaluable, where automated evaluation is infeasible or insufficient (e.g. plotting). The total number of examples is 84, of which 25 were expert evaluable, and the accuracy is 72% for the best performing model.

GPT-3 12B fine-tuned on code (Codex) is abbreviated as "davinci" and a smaller GPT-3 fine tuned on

---

* andrew.white@rochester.edu
† hockyg@nyu.edu

Prompt
```
import numpy as np

def run_mc(n_steps, T, k):
    """
    This function performs metropolis monte carlo sampling of a harmonic
    potential at temperature T and returns the trajectory
    """
    [insert]
    return traj
```

Inserted code - output (1)
```
    traj = np.zeros(n_steps)
    traj[0] = np.random.uniform(-1, 1)
    for i in range(1, n_steps):
*       x_new = np.random.uniform(-1, 1)    (2) x_new = traj[i-1] + np.random.randn()
        dE = 0.5*k*(x_new**2 - traj[i-1]**2) (3) x_new = np.random.normal(traj[i-1], 1)
        if dE < 0:
            traj[i] = x_new
        else:
            if np.random.uniform(0, 1) < np.exp(-dE/T):
                traj[i] = x_new
            else:
                traj[i] = traj[i-1]
```

FIG. 1. Example prompt and code generated for database example 'mc_harmonic'. Full output is the prompt with '[insert]' replaced by code in lower box. The asterisk indicates a line that is faulty. Inset box shows equivalent lines from two other solutions that are correct, if not necessarily optimal. This example is discussed in Sec. III A.

code (Codex) is abbreviated as "cushman." Since 2021, the models have been adjusted and the specific versions used in this work are code-cushman-001, code-davinci-002. The "incoder" models are two models from Fried et al.[17] trained on code only. We chose incoder because it is able to infill code in addition to completing code prompts, which gives a more direct comparison, and it has generally good performance. Recent benchmarks show davinci is best or nearly the best on general programming tasks.[15,24] Incoder was used as implemented in HuggingFace transformers.[25] To avoid library changes since 2021 influencing accuracy, our evaluations are done using the python version and packages from June 2021. The chosen date was based on reported training range from Ref. 24 and comes before training time of Ref. 17.

When developing example prompts and solutions, the prompts were tested and modified using davinci. Some prompt engineering was inevitable through this process.[3,26,27] However, prompts were not designed to get a correct answer and some prompts (e.g., two atom harmonic oscillator) were never correctly completed. We do emphasize that the reported accuracy is not one would expect of the first prompt constructed on-the-fly for a given problem. Rather, they are constructed to answer "how much chemistry do these LLMs know?" These figures should not be construed as upper bounds either, as recent work on prompt engineering shows that multiple steps[18] or eliciting multiple steps can further improve accuracy.[28]

Following Chen et al.,[24] a prompt completion is accurate if the code functions correctly, not if it matches a reference implementation. Most examples have both a prompt and unit tests. Expert evaluable prompts for which there are not unit tests are not reported in the accuracy, unless specified. Five completions were generated via top-k sampling[29] and multiple temperatures at $T = 0.05, 0.2, 0.5$ (softmax scaling). We explored nucleus sampling,[30] but found it to be no different than adjusting temperature for balancing diversity and correctness of completions. We chose $k = 5$ for all models, except for incoder-6B where GPU memory limitations prevented sampling more than $k = 1$. Thus, those results may be slightly inflated since accuracy is reported on only a most likely output. Error bars in all plots are 95% confidence intervals generated from bootstrap resampling across top-k.

Expert evaluation was performed on $k = 3$ outputs of davinci ($T = 0.2$, "insert" context) and accessed through a web interface.[31] Each example contains a link to a custom Google form which could be used to evaluate that example, with results saved in a spreadsheet. The multiple choice questions in the form were: "Is this question: Easy; Medium; Hard", "Is the solution: Perfect; Correct but not perfect; Runs and is almost correct; Does not run but is almost correct; Is far from correct." There was also a box for extra comments. The full set of evaluations, with personally identifiable information (student emails) removed, is available as a comma separated value (CSV) file in the Supporting Information. To make a numerical evaluation of this data in Fig. 2, we assigned scores from 1–5 with 5 being the best ("Perfect") and 1 being the worst ("Is far from correct"). To compute an overall accuracy as reported in Tab. I, we assigned "Perfect", and "Correct but not perfect" a value of 1.0, and all others 0.0, then computed the mean score for each prompt separately. It should be noted that each assessor

Topic	N	expert	incoder-6B	davinci
bio	13	2	0%	43% (0%)[1]
cheminf	10	0	20%	50%
genchem	11	0	29%	86%
md	11	3	0%	63% (81%)
plot	10	10	–	– (57%)
qm	8	3	20%	100% (59%)
sim	8	5	0%	100% (64%)
spectroscopy	11	1	30%	50%(12%)
stats	11	1	40%	70% (88%)
thermo	10	0	10%	80%
total	84[2]	23	17%	72% (57%)

TABLE I. The number of prompts by topic and best accuracy achievable in this work. "expert" is the number within a topic that must be evaluated by an expert. We used the "copyright" context for incoder-6B and "insert" for davinci and $T = 0.2$. Results are averaged across top-k sampling and/or multiple expert evaluators. [1]expert evaluator scores are in parentheses. [2]some prompts appear in multiple topics. The abbreviations of topics are biochemistry (bio), cheminformatics (cheminf), general chemistry (genchm), molecular dynamics & simulation (md), quantum mechanics (qm), methods of simulation (sim), statistics (stats), and thermodynamics (thermo).

had a different level of expertise on each topic, as well as a different level of python programming experience, although we feel all were sufficiently expert to evaluate each prompt with sufficient authority.

III. RESULTS

A. Example problem

To illustrate the kinds of tasks and impressive (if not always correct) results produced by LLMs, we show the output for one 'sim' category tasks in Fig. 1. To standardize our tasks, each tasks is phrased as a function to be filled in, as in the top box. This prompt includes a first line which loads the numerical python (numpy[32]) library, which gives additional 'context' (see below). The rest of the information for the LLM is contained in two places, the names of the variables given as inputs 'n_steps', 'T', 'k', and a comment string which says what the function does/should do. In this case, the function should perform Metropolis Monte Carlo for a harmonic potential. Implicit in the instruction by the creator is that k represents the spring constant, and so this code should produce samples from the energy function $U(x) = 1/2k(x - x_0)^2$, with $x_0 = 0$ since it was not specified as an input, and also that reduced units are used, such that Boltzmann's constant $k_B = 1.0$. We can see that—with quite minimal instruction—the code in the output is correct except for an error on the line indicated with a '*'; in this line, the position of the particle is completely resampled from scratch on the range [-1,1). This code would actually be fine if the system were constrained to be within a box of length 2, and in the limit of $k >> 1$ it will also appear

to give correct results. The inset shows the equivalent line in two other outputs of the model, both of which are acceptable; one displaces the position by a Gaussian random number with $\mu = 0$ and $\sigma^2 = 1$, and the second chooses a new position from a Gaussian with mean centered at the current position and $\sigma^2 = 1$. Note that neither of these is optimized for the choice of (k,T), as $\sigma^2 = 1$ may be too large or too small to be efficient, depending on the spring constant and temperature. Finally, in one of the other two outputs for this example (available in the SI or on the results website), k is interpreted as Boltzmann's constant, and the harmonic system is given a spring constant of 1.0 implicitly; this is a reasonable result of the model. It illustrates how the author must be careful about what is implicit in their prompt and what is stated explicitly (e.g. here, that T is temperature).

B. Expert evaluations

Davinci, the best performing model, does have broad knowledge of equations and common calculations across multiple domains of chemistry. Table I gives the overall accuracy across the topics, models, and expert evaluable topics. Both models can correctly answer prompts across a range of topics, with davinci doing best. About 30 percentage points of accuracy is from prompt engineering, which is discussed further below.

On average, the accuracy on human evaluable topics is lower, reflecting their increased difficulty. These prompts include tasks like writing an input file for NWChem,[33] implementing a Monte Carlo simulation of a harmonic oscillator (Fig. 1), and generating a complex multi-panel plot. Fig. 2 shows a breakdown of difficulty from the individual evaluations. There is a balance of easy and hard prompts in the dataset, as judged by experts. Our primary result here is that the accuracy of the model is negatively correlated with perceived prompt difficulty, as
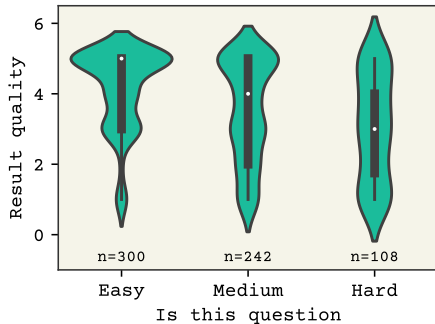

FIG. 2. 650 evaluations of davinci completions by the nine coauthors who are postdoctoral scholars or Ph.D. students in chemistry or chemical engineering. Scoring is described in Sec. II. We find that the typical result quality (white dot) drops from 'Perfect,' to 'Correct but not perfect', to 'Runs and is almost correct' as perceived difficulty increased.

might be expected but did not necessarily have to be the case. We did not do any randomization or controls; each evaluator was able to see all prompts and all outputs, and so we acknowledge that scores could be biased by factors such as the order of the prompts on the website, and the order that results for a given prompt were presented on the website. In the rest of this article, we focus only on prompts whose correctness can be evaluated by comparison with an expected solution in an automated fashion.

C. How to improve performance

There is a large accuracy gain when using basic prompt engineering strategies. Fig. 3 shows the effect of different "contexts" on accuracy across models. A context here is code prepended before all prompts, or all prompts within a topic. The contexts are given both in the Supporting Information and our accompanying code. "Custom" includes two pieces: some imports related to the topic (e.g., `rdkit`[34] for cheminf) and a single example to teach the model how to indicate the end of a prompt completion. The imports are not just to prevent import errors — they influence the completions and give context. For example, a "structure" after importing `rdkit` means a bonded arrangement of atoms; in contrast, a structure after importing `openmm`[35] (a molecular dynamics simulation code) would implicitly mean a 3D arrangement of atoms, e.g. obtained from a PDB file.

The completion example is a one line statement (e.g., printing version number of imported package) with a comment above and `#end` below. This causes the LLM to end completions with `#end`. We tried to ad-hoc look for certain keywords such as new function `def`s, `return`s, or comments as completion ends, but these heuristics were often violated. The completion example is significant for the cushman model, which can only do completions but not insertions. For davinci and the incoder models, we can replace this with the "insert" contexts which have the same imports but use a model capability to infill at a special insert token (as in Fig. 1). Avoiding our completion example in the context seems to be insignificant for davinci, but important for incoder.

LLMs seem to be very very very susceptible to conditioning contexts like adding the word "very" many times to improve a completion[36] or stating that the code "has no bugs." We explored this in our benchmarks in two ways. We tried inserting copyright notices and found in Figs. 3 and Fig. 4 that it does significantly improve accuracy at higher temperatures. This makes intuitive sense; lowering temperature makes the LLM choose more likely completions and a copyright notice would more often be included with standard/quality code, thus giving a similar effect to lowering temperature. The best performing model/temperature combination was not improved because it already had a low temperature. We also tried inserting the statement "This is written by an expert Python programmer" as suggested by Austin[37], and saw slightly less improvement. Similar recent work has found context or specific phrases (e.g., "let's think step by step") can give large accuracy improvements.[28,38] Fried et al.[17] have recently explored using metadata, including popularity of code, as a mechanism to condition completions, so that we do not need to use ad-hoc prompt engineering.

Aside from contexts, there are a few strategies to ensure a prompt aligns the intent of a user with the completion. If the prompt contains programming mistakes or spelling mistakes, then the completion will be of similar quality. So a correctly spelled and intelligible prompt is necessary.

The LLM tries to agree with each word in the prompt. If a prompt is a function declaration and uses the phrase "compute the moment", the model will probably not return the value. Thus, the word "return" should be used. If a package is imported in the prompt, the model will try to make use of it. This can lead to problems if many packages are imported – it can be unexpected as to which packages the model will use, or, the model thinks it must use all of them.

A major source of the errors in some of the categories such as 'md' is the proper use of functions from a package such as `mdtraj`, in particular, improper knowledge of how many and what type of values are returned by that function; this could be a simple error or due to training on an earlier version of the module; these results may be able to be improved in the future by 'fine tuning' the LLM on examples from a particular package that is frequently used in one's work, or by adding additional context.

D. Molecular structures

Our goal is to evaluate how much chemistry LLMs know. Besides evaluating tasks that can be expressed as programs, we also explored whether LLMs can connect natural language directly with molecular structures. We tested both InstructGPT[39] and Codex (davinci) in these examples, but found InstructGPT to work better. Neither could convert from molecular SMILES to name of molecule, as demonstrated with 0% accuracy on 100 random molecules from pubchem[40] when we tried SMILES length of less than 60 characters (relatively small/simple molecules). The attempt from InstructGPT is shown in the Supporting Information. InstructGPT was able to convert a sentence describing a molecule into SMILES, as shown with examples in Fig. 5. InstructGPT is able to connect functional groups from SMILES to natural language. It is also able to correlate molecular properties like lipophilicity with SMILES. InstructGPT rarely generates invalid SMILES; only the first molecule in Fig. 5 had a single invalid character (see Supporting Information for SMILES). It appears that InstructGPT or other LLMs could be trained/fine-tuned on the connection between natural language and chemical structures.
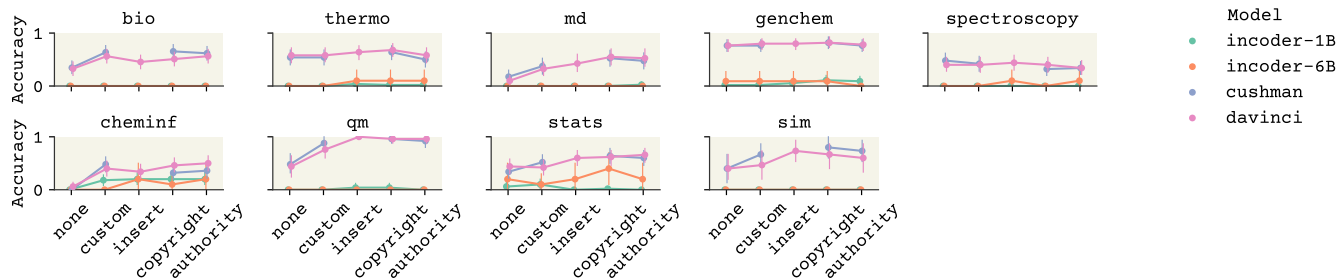
FIG. 3. A comparison of accuracy of the LLMs compared in this study across different contexts. Adding context – short comments/imports – generally improves accuracy across topic and model. Error bars are 95% confidence intervals from bootstrapping across individual prompts, temperature, and from multiple completions.
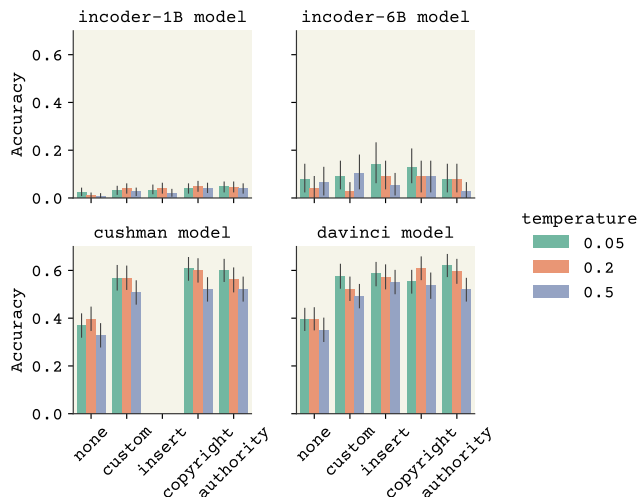

FIG. 4. Comparison of context effect across models and temperatures. Having a custom context is most important. Note that insert, copyright, and authority include the "custom" context. Error bars are 95% confidence intervals from bootstrapping across individual prompts, temperature, and from multiple completions. Cushman cannot do insertions.

FIG. 5. Generating molecules with InstructGPT (text-davinci-002). Prompts are shown in annotations. The strongly lipophilic molecule is C_{505}, a polystyrene that is indeed strongly lipophilic. Most examples contain mistakes, but were mostly valid. The top-left example had an ambiguous ring indicator index which was removed prior to drawing.

E. Discussion

Davinci seems to not reason well about computational chemistry. If we prompt davinci to use to a "highly accurate single-point" quantum calculation in `pyscf`,[41] it will frequently use relativistic Hartree-Fock regardless of the property being computed because it has memorized that "relativistic" is associated with accurate. Another example is in the "force constant" prompt which is meant to compute the force constant for a two-atom harmonic oscillator with different masses given a wavelength. Perhaps because this is an unusual variant of a common question (converting between force constant and wavelength), davinci always fails on this question and is unable to rearrange the equation to take a harmonic mean of masses.
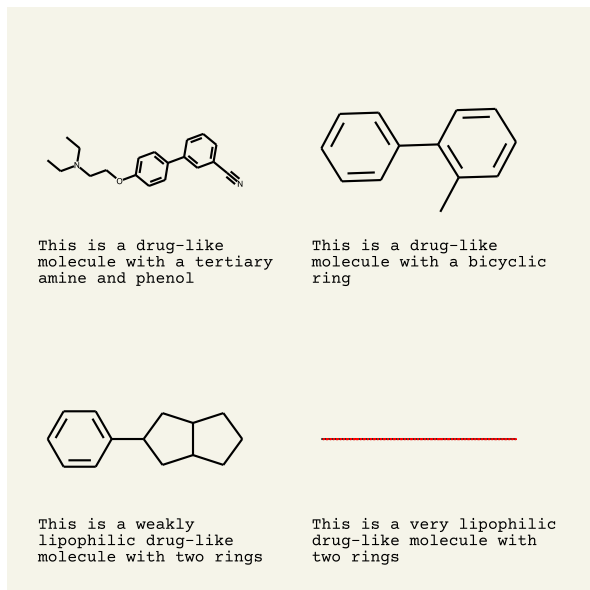
Davinci may also hallucinate functions that do not exist. If a difficult prompt is given, for example "return the residual dipole couplings given a SMILES string," the model will simply try to use a non-existent method `MolToRDC`. As reported previously,[21] LLMs are not able to do chemical reasoning when completing prompts.

Finally, we'd like to anecdotally note that the LLMs could perform many of the benchmark problems if the natural language was in Chinese, German, or Spanish. We did not explore this in depth, but a few example prompts written in Mandarin can be found in the Supporting Information. The use of LLMs with prompts that are not in English may be a valuable tool for lowering the barrier to employing computational tools for those who are not native English speakers, and who therefore may have a harder time interpreting documentation and pro-

gramming forums.

IV. CONCLUSIONS

LLMs are now easily available via tools like tabnine[42] or copilot.[43] We've found high accuracy on computational chemistry questions, and it is inevitable that students and researchers will begin using these tools. From our results, high accuracy should be expected with reasonable prompts. Tricks like inserting copyright notices at the top of a source file seems to be another way to improve accuracy. We found that humans are able to gauge accuracy for easy to medium prompts, but care should be taken if using completions of difficult prompts. The seeming inability to generate syntactically invalid code means LLMs often produce *something*, but it is up to the user to assess it. We also found somewhat unexpected capabilities like generating molecules from natural language and accurate completions with non-English prompts. For a broader discussion of what impact this will have on education, we refer the interested reader to our earlier perspective article.[5]

AUTHOR CONTRIBUTIONS

ADW and GMH wrote NLCC software and designed nlcc-database, website, and human evaluation form. They contributed examples to the nlcc-data repository and performed data analysis. All other authors contributed examples to the nlcc-data repository, participated in the expert evaluation, and assisted in writing the manuscript.

V. SUPPORTING INFORMATION

Supporting figures, tables, and text are included in the Supporting Information. Accuracy data is available as comma separated value files. Contexts are available as a markup file. The completions as HTML presented to expert evaluators is available at `DOI: 10.5281/zenodo.6800475`.

---

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, Attention is all you need, Adv. Neural Inf. Process Syst. **30** (2017).

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805 (2018).

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, Language models are few-shot learners, Adv. Neural Inf. Process Syst. **33**, 1877 (2020).

[4] A. Srivastava, A. Rastogi, A. Rao, A. A. M. Shoeb, A. Abid, A. Fisch, A. R. Brown, A. Santoro, A. Gupta, A. Garriga-Alonso, *et al.*, Beyond the imitation game: Quantifying and extrapolating the capabilities of language models, arXiv preprint arXiv:2206.04615 (2022).

[5] G. M. Hocky and A. D. White, Natural language processing models that automate programming will transform chemistry research and teaching, Digital Discovery **1**, 79 (2022).

[6] S. Wang, Y. Guo, Y. Wang, H. Sun, and J. Huang, Smiles-bert: large scale unsupervised pre-training for molecular property prediction, in *Proceedings of the 10th ACM international conference on bioinformatics, computational biology and health informatics* (2019) pp. 429–436.

[7] N. Frey, R. Soklaski, S. Axelrod, S. Samsi, R. Gomez-Bombarelli, C. Coley, and V. Gadepally, Neural scaling of deep chemical models, ChemRxiv 10.26434/chemrxiv-2022-3s512 (2022).

[8] D. Flam-Shepherd, K. Zhu, and A. Aspuru-Guzik, Language models can learn complex molecular distributions, Nat. Commun. **13**, 1 (2022).

[9] J. Ross, B. Belgodere, V. Chenthamarakshan, I. Padhi, Y. Mroueh, and P. Das, Do large scale molecular language representations capture important structural information?, arXiv preprint arXiv:2106.09553 (2021).

[10] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, *et al.*, Exploring the limits of transfer learning with a unified text-to-text transformer., J. Mach. Learn. Res. **21**, 1 (2020).

[11] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, *et al.*, The pile: An 800gb dataset of diverse text for language modeling, arXiv preprint arXiv:2101.00027 (2020).

[12] D. Weininger, Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules, J. Chem. Inf. Comput. Sci. **28**, 31 (1988).

[13] C. Nantasenamat, "would be cool to have gpt-3 generate new chemical structures in smiles notation?", Twitter ,

1516794237391863810 (2022).

[14] A. D. White, "as suggested by @thedataprof, gpt-3 can actually generate molecules. very clever idea! prompt was "the smiles for this drug-like molecular are:", Twitter , 1516795519284228106 (2022).

[15] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, A systematic evaluation of large language models of code, in *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming* (2022) pp. 1–10.

[16] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, *et al.*, Program synthesis with large language models, arXiv preprint arXiv:2108.07732 (2021).

[17] D. Fried, A. Aghajanyan, J. Lin, S. Wang, E. Wallace, F. Shi, R. Zhong, W.-t. Yih, L. Zettlemoyer, and M. Lewis, Incoder: A generative model for code infilling and synthesis, arXiv preprint arXiv:2204.05999 (2022).

[18] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, A conversational paradigm for program synthesis, arXiv preprint (2022).

[19] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, Language models are unsupervised multitask learners, OpenAI blog **1**, 9 (2019).

[20] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, *et al.*, Codebert: A pretrained model for programming and natural languages, arXiv preprint arXiv:2002.08155 (2020).

[21] E. M. Bender and A. Koller, Climbing towards nlu: On meaning, form, and understanding in the age of data, in *Proceedings of the 58th annual meeting of the association for computational linguistics* (2020) pp. 5185–5198.

[22] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, On the dangers of stochastic parrots: Can language models be too big? 🦜, in *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency* (2021) pp. 610–623.

[23] `https://github.com/ur-whitelab/nlcc-data`.

[24] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, *et al.*, Evaluating large language models trained on code, arXiv preprint arXiv:2107.03374 (2021).

[25] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, *et al.*, Huggingface's transformers: State-of-the-art natural language processing, arXiv preprint arXiv:1910.03771 (2019).

[26] S. H. Bach, V. Sanh, Z.-X. Yong, A. Webson, C. Raffel, N. V. Nayak, A. Sharma, T. Kim, M. S. Bari, T. Fevry, *et al.*, Promptsource: An integrated development environment and repository for natural language prompts, arXiv preprint arXiv:2202.01279 (2022).

[27] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le, and D. Zhou, Chain of thought prompting elicits reasoning in large language models, arXiv preprint arXiv:2201.11903 (2022).

[28] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, Large language models are zero-shot reasoners, arXiv preprint arXiv:2205.11916 (2022).

[29] A. Fan, M. Lewis, and Y. Dauphin, Hierarchical neural story generation, arXiv preprint arXiv:1805.04833 (2018).

[30] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, The curious case of neural text degeneration, arXiv preprint arXiv:1904.09751 (2019).

[31] `https://ur-whitelab.github.io/nlcc-data/`.

[32] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, *et al.*, Array programming with numpy, Nature **585**, 357 (2020).

[33] M. Valiev, E. J. Bylaska, N. Govind, K. Kowalski, T. P. Straatsma, H. J. J. Van Dam, D. Wang, J. Nieplocha, E. Aprà, T. L. Windus, *et al.*, Nwchem: A comprehensive and scalable open-source solution for large scale molecular simulations, Comp. Phys. Comm. **181**, 1477 (2010).

[34] G. Landrum *et al.*, Rdkit: A software suite for cheminformatics, computational chemistry, and predictive modeling, Greg Landrum (2013).

[35] P. Eastman, J. Swails, J. D. Chodera, R. T. McGibbon, Y. Zhao, K. A. Beauchamp, L.-P. Wang, A. C. Simmonett, M. P. Harrigan, C. D. Stern, *et al.*, Openmm 7: Rapid development of high performance algorithms for molecular dynamics, PLoS Comp. Biol. **13**, e1005659 (2017).

[36] P. Isola, "language-conditional models can act a bit like decision transformers, in that you can prompt them with a desired level of "reward". e.g., want prettier #dalle creations? "just ask" by adding "[very]^n beautiful":", Twitter , 1532189616106881027 (2022).

[37] J. Austin, "we found that code models get better when you prompt them with i'm an expert python programmer. the new anthropic paper did something similar, prefixing the model's response with i've tested this function myself so i know that it's correct:, Twitter , 1515063524258627586 (2022).

[38] Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan, *et al.*, Training a helpful and harmless assistant with reinforcement learning from human feedback, arXiv preprint arXiv:2204.05862 (2022).

[39] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, *et al.*, Training language models to follow instructions with human feedback, arXiv preprint arXiv:2203.02155 (2022).

[40] S. Kim, J. Chen, T. Cheng, A. Gindulyte, J. He, S. He, Q. Li, B. A. Shoemaker, P. A. Thiessen, B. Yu, *et al.*, Pubchem 2019 update: improved access to chemical data, Nucleic Acids Res. **47**, D1102 (2019).

[41] Q. Sun, T. C. Berkelbach, N. S. Blunt, G. H. Booth, S. Guo, Z. Li, J. Liu, J. D. McClain, E. R. Sayfutyarova, S. Sharma, *et al.*, Pyscf: the python-based simulations of chemistry framework, Wiley Interdiscip. Rev. Comput. Mol. Sci. **8**, e1340 (2018).

[42] `https://www.tabnine.com/`.

[43] `https://github.com/features/copilot`.