Digital Discovery

ROYAL SOCIETY OF CHEMISTRY

PAPER

View Article Online
View Journal | View Issue

Cite this: *Digital Discovery*, 2023, **2**, 368

Assessment of chemistry knowledge in large language models that generate code†

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

In this work, we investigate the question: do code-generating large language models know chemistry? Our results indicate, mostly yes. To evaluate this, we introduce an expandable framework for evaluating chemistry knowledge in these models, through prompting models to solve chemistry problems posed as coding tasks. To do so, 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 that 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. Our 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.

Received 17th August 2022
Accepted 19th January 2023

DOI: 10.1039/d2dd00087c

rsc.li/digitaldiscovery

I. Introduction

Large language models (LLMs) are multi-billion parameter transformer neural networks[1] that are trained on enormous collections of documents (a 'corpus') without supervision or labels.[2] LLMs can perform 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] On Nov 30, 2022, OpenAI released an interactive interface to an LLM termed ChatGPT (ref. 10) which substantially increased interest in this area as well as use by scientists for coding and writing tasks. An open question for LLMs such as GPT-3,[3] T5,[11] or GPT-neo (ref. 12) 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 that relationships between SMILES and natural language is possible with GPT-3. SMILES is the standard method of representing molecules as strings.[13] It is even possible to loosely edit structures *via* natural language (see Fig. 6).[14,15] 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 in ref. 16. Here, we focus our study on whether LLMs that generate code[17] 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[17–19]—a user provides a sequence of text (called the prompt) and it proposes a continuation of the text (the completion).[20] There are LLMs trained on code that can infill or match encoder/decoder natural language to code like Code-

[a]*Department of Chemical Engineering, University of Rochester, USA. E-mail: andrew.white@rochester.edu*
[b]*Vial Health Technology, Inc., USA*
[c]*Department of Chemistry, New York University, USA. E-mail: hockyg@nyu.edu*
[d]*Simons Center for Computational Physical Chemistry, New York University, USA*
[e]*Department of Chemistry, University of Rochester, USA*

† Electronic supplementary information (ESI) available: Supporting figures, tables, and text. Accuracy data are available as comma separated value files. Contexts are available as a markup file. The responses from the model (completions) which were the basis for expert evaluators are available in HTML format at **https://doi.org/10.5281/zenodo.6800475**. See DOI: **https://doi.org/10.1039/d2dd00087c**

BERT,[21] but they are typically used for embedding code for tasks like classification, document retrieval, or translating code to natural language. Because it is not reasonable to use encoder–decoder or encoder-only models to generate code or answer questions with open-ended length, this paper explores solely decoder-only causal language models.

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.[22,23] 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.[24] To generate these problems, we first decided upon a list of categories of chemistry and chemical engineering knowledge, listed in Table 1, and set a goal of having at least 10 examples in each category for our initial database of problems. Members of our research groups (the authors of this paper), who we consider to have sufficient expertise in these areas due to formal schooling, research, and teaching experience, contributed the prompts and reference solutions for these categories.

The examples in this table span a range of topics that we consider common questions across chemistry fields. There is some representation of computational chemistry research topics (categories corresponding to performing chemical simulations (sim), analyzing molecular dynamics simulations (md), chemical informatics (cheminf), and some quantum mechanics (qm)), but this constitutes less than half of the initial prompts created by us. The rest correspond to typical questions that one might encounter in general chemistry (genchem), biochemistry (bio), physical chemistry (thermodynamics, quantum mechanics, and spectroscopy), and in laboratory classes (plotting and statistics).

Within this set of topics, 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 75% for the best performing model.

There is a strong correlation between the model parameter count and accuracy,[25] so we focus only on the largest models with more than 1B parameters. The architectures of models are all decoder-only like GPT-3 (ref. 3) with the ability to insert completions,[26] (except when noted). The first model is a GPT-3 12B fine-tuned on code (Codex) abbreviated as "cushman". It is known as code-cushman-001 in the OpenAI API.[27] This is modified from the original one in Austin *et al.*[17] somewhat and is described as "a stronger, multilingual version of the Codex 12B model.".[28] We also used code-davinci-002, abbreviated as "davinci". This model is part of the category of "GPT-3.5" models that are derived from GPT-3.[29] The number of parameters in davinci-class models is not public information, but may match the 175B parameters of the model described in the GPT-3.5 paper.[30] We also considered the recent text-davinci-003 model which is derived from code-davinci-002 with a reinforcement-learning adaption from human user feedback[30] – although this model became available only after human evaluation (below) was complete, so our analysis is reported only on automated evaluations. This model is denoted as 'davinci3' here. Finally, from publicly available information we know that ChatGPT is based on a slightly modified version of GPT-3.5, and so we expect its performance to be comparable to that of the model; however, it does not have an API that would allow us to systematically probe any differences in our study. One example use of ChatGPT is given in the ESI.†

We also study two "incoder" models from Fried *et al.*[18] 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. Lastly, we consider the 'codegen' model,[31] which is another decoder-only model trained on a similar dataset to 'incoder'. It was not trained for infilling, because it was designed for back-and-forth code synthesis with natural language. Although it is not exactly analogous to the other models, it is one of very few competitive models that can generate working code, and so we include it here for comparison.[31]

Recent benchmarks show that davinci is the best or nearly the best for general programming tasks.[16,32] Incoder was used as implemented in HuggingFace transformers.[33] To avoid library changes since 2021 influencing the accuracy, our evaluations are performed using the python version and packages from June 2021. The chosen date was based on the reported training range from ref. 32 and comes before the training time in ref. 18.

Table 1 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, "authority" for codegen-16B, and "insert" for davinci and $T = 0.2$ (best for all models). Accuracies are averaged (macro-averaging) across top-k sampling (we consider correct if valid prompt appeared in top-k results). Expert accuracies are macro-averaged across topics/prompts

Topic	N	Expert	Incoder	Codegen	Davinci		Davinci3
Bio	13	2	0%	29%	43%	(0%)[a]	86%
Cheminf	10	0	20%	20%	50%		50%
Genchem	11	0	29%	86%	86%		86%
md	11	3	0%	13%	63%	(81%)	88%
Plot	10	10	—	—	—	(57%)	—
qm	8	3	20%	60%	100%	(59%)	100%
sim	8	5	0%	0%	100%	(64%)	100%
spect	11	1	30%	20%	50%	(12%)	40%
stats	11	1	40%	80%	70%	(88%)	60%
Thermo	10	0	10%	10%	80%		70%
Total	84[b]	23	17%	35%	72%	(57%)	75%

[a] Expert evaluator scores are in parentheses. [b] 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), spectroscopy (spect), statistics (stats), and thermodynamics (thermo)

When developing example prompts and solutions, the prompts were tested and modified using davinci. Some prompt engineering was inevitable through this process.[3,34,35] 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 what 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 (sometimes known as using "scratchpads")[19] or eliciting multiple steps can further improve accuracy.[29]

Following Chen *et al.*,[32] 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. The accuracy of expert evaluable prompts for which there are no unit tests is not reported, unless specified. Five completions were generated *via* top-k sampling[36] and multiple temperatures at $T = 0.05, 0.2, 0.5$ (softmax scaling). We explored nucleus sampling,[37] but found it to be no different compared to adjusting the temperature for balancing the 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, these 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.[38] 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. This evaluation did breakout more detailed information like alignment between the prompt and completion or hazards of completion, similar to that recently proposed by Khlaaf.[39] The full set of evaluations, with personally identifiable information (student emails) removed, is available as a comma separated value (CSV) file in the ESI.† To make a numerical evaluation of this data as shown in Fig. 3, 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 Table 1, we assigned "Perfect", and "Correct but not perfect" a value of 1.0, and all others 0.0, and then computed the mean score for each prompt separately. It should be noted that each assessor 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 problems

To illustrate the kinds of tasks and impressive (if not always correct) results produced by LLMs, we show the output for one 'sim' category task in Fig. 1. To standardize our tasks, each task 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[40]) 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

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)
        dE = 0.5*k*(x_new**2 - traj[i-1]**2)
        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]
```

(2) `x_new = traj[i-1] + np.random.randn()`
(3) `x_new = np.random.normal(traj[i-1], 1)`

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

energy function $U(x) = \frac{1}{2}k(x - x_0)^2$, 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 in 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 $k \gg 1$ 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 the 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 ESI† or on the result 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 inference 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 the temperature).

Fig. 2 shows an additional example to highlight how the davinci-codex model internally contains knowledge of chemistry topics (in this case, general chemistry pertaining to phase equilibrium). The output shows that the model "knows" the relevant rearrangement of the Claussius–Clapeyron equation, and returns the appropriate result, assuming that the heat of vaporization ('Hvap') was given in joules mol^{-1}. One figure in the ESI† shows that we can use ChatGPT to solve the same problem, either by asking it to fill in the "[insert]" text with the correct solution, or by describing the problem conversationally. When an API for ChatGPT is available, we would expect the performance in the former mode to be very similar to that of the underlying GPT-3.5 model.

Prompt

```
import math
import sys

def claussius(HVap, T1, P1, T2):
    """
    This function returns the phase
    transition pressure at temperature T2
    given a heat of vaporization HVap,
    and and reference temperature and
    pressure T1 and P1
    """
    [insert]
    return P2
```

Inserted code - output (4)

```
    P2 = P1*math.exp((HVap/8.314)*
                     ((1/T1)-(1/T2)))
```

Fig. 2 Example prompt and code generated for the database example 'claussius'. Full output is the prompt with '[insert]' replaced by code in the lower box. Davinci passed our automated check for this example on three out of five tries.

B. Expert evaluations

Davinci, the best performing model, does have broad knowledge of equations and common calculations across multiple domains of chemistry. Table 1 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 performing the best. About 30 percentage points of accuracy are from prompt engineering, which is discussed further below.

On average, the accuracy for human evaluable topics is lower, reflecting their increased difficulty. These prompts include tasks like writing an input file for NWChem,[41] implementing a Monte Carlo simulation of a harmonic oscillator (Fig. 1), and generating a complex multi-panel plot. Fig. 3 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 might be expected but did not necessarily have to be the case. We did not perform 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. 4 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 ESI† and our accompanying code. "Custom" includes two pieces: some imports related to the topic (*e.g.*, rdkit[42] for cheminf) and a single example to teach the model how to indicate the end of a prompt completion. The

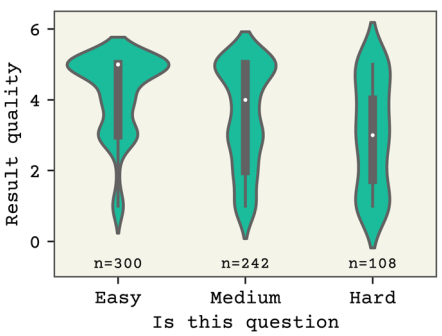

Fig. 3 650 evaluations of davinci completions by the nine coauthors who are postdoctoral scholars or PhD students in chemistry or chemical engineering. Scoring is described in Sec. II. We find that the typical result quality (white dot) drops from 'Perfect,' to 'Correct but not perfect', to 'Runs and is almost correct' as perceived difficulty increased.

imports are not just to prevent errors due to failure to include relevant libraries—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[43] (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 the version number of an 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 defs, returns, or comments as completion ends, but these heuristics were often violated. The completion example is significant for the Cushman model, which can only perform completions but not insertions. For the davinci and 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 davincni, but important for incoder.

LLMs seem to be very susceptible to conditioning contexts, like adding the word "very" many times to improve a completion[44] or stating that the code "has no bugs". We explored this in our benchmarks in two ways. We tried inserting copyright notices and found, as shown in Fig. 4 and 5 that it does significantly improve accuracy at higher temperatures. This makes intuitive sense; lowering the 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 the 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,[45] and saw slightly less improvement. A similar recent work has found context or specific phrases (*e.g.*, "let's think step by step") that elicit chain-of-thought outputs which can give large accuracy improvements.[29,46] Fried *et al.*[18] and Wei *et al.*[35] 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. Interestingly, the results from davinci3 show that the improvements to the NLP model through human feedback removed some of the observed sensitivity[30] to prompt engineering on our examples.

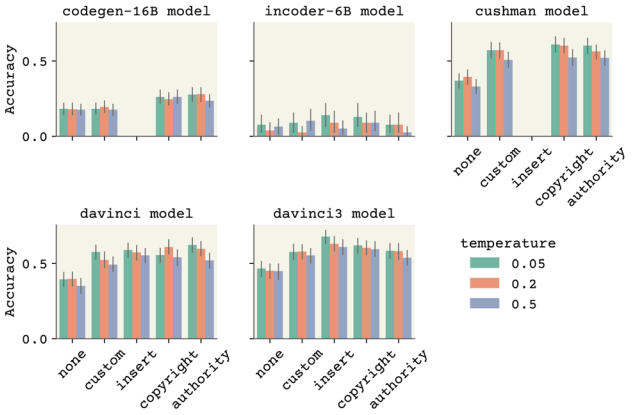

Fig. 5 Comparison of the context effect across models and temperatures. Having a custom context is most important. Note that insert, copyright, and authority include the "custom" context. Error bars are 95% confidence intervals from bootstrapping across individual prompts and temperatures, and from multiple completions. Cushman and codegen cannot perform insertions.

Aside from contexts, there are a few strategies to ensure that 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 if 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 improper 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

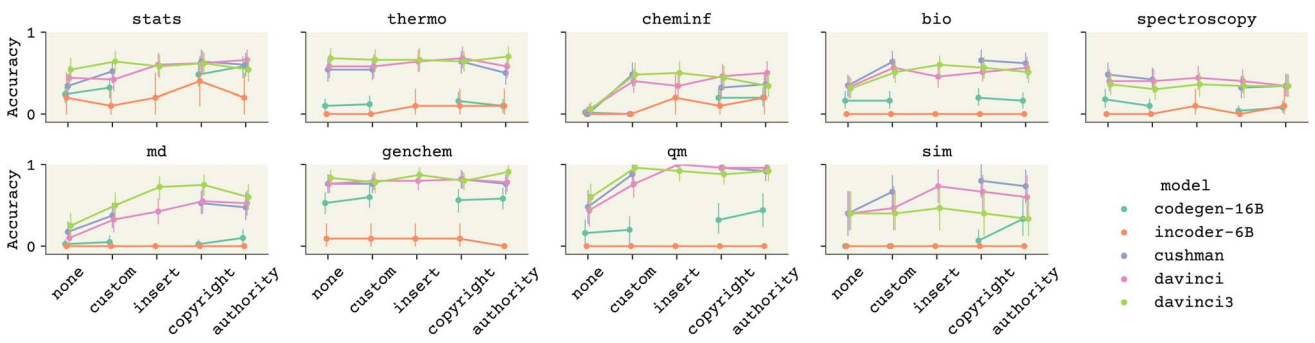

Fig. 4 A comparison of accuracy of the LLMs compared in this study across different contexts, broken down by category. Adding context – short comments/imports – generally improves accuracy across topics and models. Error bars are 95% confidence intervals from bootstrapping across individual prompts and temperatures, and from multiple completions.

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[30] and davinci in these examples, but found InstructGPT to work better. Neither could convert from molecular SMILES to the name of the molecule, as demonstrated with 0% accuracy on 100 random molecules from pubchem[47] when we tried a SMILES length of less than 60 characters (relatively small/simple molecules). The attempt from InstructGPT is shown in the ESI.† InstructGPT was able to convert a sentence describing a molecule into SMILES, as shown with examples in Fig. 6. InstructGPT is able to connect functional groups from SMILES to natural language. The molecules are not exact matches, but there is some correlation (*e.g.*, oxygen near a ring for phenol and amine). It is also able to correlate molecular properties like lipophilicity with SMILES. InstructGPT rarely generates invalid SMILES; only the first molecule in Fig. 6 had a single invalid character (see the ESI† for SMILES). It appears that InstructGPT or other LLMs could be trained/fine-tuned on the connection between natural language and chemical structures. Recently, specific models that can translate between molecular structure and natural language have also been trained from scratch.[48]

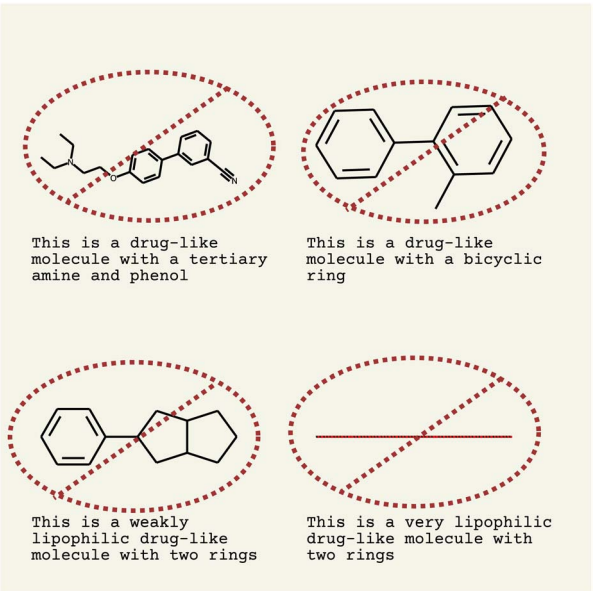

Fig. 6 Generating molecules with InstructGPT (text-davinci-002). Prompts are shown in annotations. The strongly lipophilic molecule is C_{505}, a polystyrene that is indeed strongly lipophilic. Most examples contain mistakes, but were mostly valid. The top-left example had an ambiguous ring indicator index which was removed prior to drawing. All structures do not match the prompt exactly (indicated by a crossed-icon), but do have details correlated with the prompt.

E. Discussion

Davinci seems to not reason well about computational chemistry. If we prompt davinci to use a "highly accurate single-point" quantum calculation in pyscf,[49] 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 the 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,[22] LLMs are not able to perform chemical reasoning when completing prompts.

We would 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 ESI.† The use of LLMs with prompts that are not in English may be a valuable tool for lowering the barrier for employing computational tools for those who are not native English speakers, and who therefore may have a harder time interpreting documentation and programming forums.

IV. Conclusions

LLMs are now easily available *via* tools like tabnine,[50] copilot,[51] or ChatGPT.[52] We have found high accuracy on 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. We emphasize that our results only give lower bounds on the chemistry knowledge in these models, since they cover only the specific topics so far included in our database, and further prompt engineering or other strategies for evaluating this knowledge besides python function writing could elicit even better results.

Tricks like inserting copyright notices at the top of a source file seems to be another way to improve accuracy, although fine-tuning with human feedback mitigates this effect,[30] as seen in davinci3. 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 ability to always generate syntactically valid 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 interested readers to our earlier perspective article.[5]

Data availability

Accuracy data is available as comma separated value files in the ESI.† Contexts are available as a markup file included in the ESI.† The responses from the model (completions) which were the basis for expert evaluators are available in HTML format at **https://doi.org/10.5281/zenodo.6800475**. Code used to create completions with contexts is available at **https://github.com/whitead/nlcc**. Incoder model is available at **https://github.com/dpfried/incoder/blob/main/README.md**. OpenAI Codex requires an access key to use and its model and analysis are discussed in **https://arxiv.org/abs/2107.03374**.

Author contributions

A. D. W. and G. M. H. wrote NLCC software and designed the nlcc-database, website, and human evaluation form. They contributed examples to the nlcc-data repository, performed data analysis, and drafted the manuscript. All other authors contributed examples to the nlcc-data repository, participated in the expert evaluation, and assisted in writing the manuscript.

Conflicts of interest

After submission of this manuscript, A. D. W. worked as a paid consultant for OpenAI, the developers of some of the models presented in this work.

Acknowledgements

Research reported in this work was supported by the National Institute of General Medical Sciences of the National Institutes of Health under award number R35GM137966 (to A. D. W.) and R35GM138312 (to G. M. H.). HAG was supported by NSF award 1751471. MA, SC, and Z. Y. were supported by NIH award R35GM137966. G. P. W. was supported by NSF award 1764415. S. S. and Y. S. were partially supported by NIH award R35GM138312, WJPC by R35GM138312-02S1, and K. L. partially by Department of Energy award DESC0020464. S. S. and K. L. were also partially supported by the Simons Foundation Grant No. 839534. We thank Drs Sanjib Paul, David Gomez, and Navneeth Gokul who also contributed some examples to the repository.

References

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

2 J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, Bert: pre-training of deep bidirectional transformers for language understanding, *arXiv*, 2018, preprint, arXiv:1810.04805, DOI: **10.48550/arXiv.1810.04805**.

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.*, 2020, **33**, 1877.

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*, 2022, preprint, arXiv:2206.04615, DOI: **10.48550/arXiv.2206.04615**.

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

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*, 2022, preprint, DOI: **10.26434/chemrxiv-2022-3s512**.

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

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*, 2021, preprint, arXiv:2106.09553, DOI: **10.48550/arXiv.2106.09553**.

10 **https://openai.com/blog/chatgpt/**.

11 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.*, 2020, **21**, 1.

12 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 800 gb dataset of diverse text for language modeling, *arXiv*, 2020, preprint, arXiv:2101.00027, DOI: **10.48550/arXiv.2101.00027**.

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

14 C. Nantasenamat, "would be cool to have gpt-3 generate new chemical structures in smiles notation?", Twitter, 1516794237391863810, 2022 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 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 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.

15 C. Nantasenamat, "would be cool to have gpt-3 generate new chemical structures in smiles notation?", Twitter, 1516794237391863810, 2022 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 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 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.

16 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.

17 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*, 2021, preprint, arXiv:2108.07732, DOI: **10.1145/3520312.3534862**.

18 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*, 2022, preprint, arXiv:2204.05999, DOI: **10.48550/arXiv.2204.05999**.

19 E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese and C. Xiong, A conversational paradigm for program synthesis, *arXiv*, 2022, preprint, arXiv:2203.13474, DOI: **10.48550/arXiv.2203.13474**.

20 A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, Language models are unsupervised multitask learners, *OpenAI blog*, 2019, vol. 1, p. 9.

21 Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, *et al.*, Codebert: A pre-trained model for programming and natural languages, *arXiv*, 2020, preprint, arXiv:2002.08155, DOI: **10.48550/arXiv.2002.08155**.

22 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.

23 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.

24 **https://github.com/ur-whitelab/nlcc-data**.

25 P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar, *et al.*, Holistic evaluation of language models, *arXiv*, 2022, preprint, arXiv:2211.09110, DOI: **10.48550/arXiv.2211.09110**.

26 M. Bavarian, H. Jun, N. Tezak, J. Schulman, C. McLeavey, J. Tworek and M. Chen, Efficient training of language models to fill in the middle, *arXiv*, 2022, preprint, arXiv:2207.14255, DOI: **10.48550/arXiv.2207.14255**.

27 **https://Openai.com**.

28 **https://beta.openai.com/docs/model-index-for-researchers**.

29 T. Kojima, S. S. Gu, M. Reid, Y. Matsuo and Y. Iwasawa, Large language models are zero-shot reasoners, *arXiv*, 2022, preprint, arXiv:2205.11916, DOI: **10.48550/arXiv.2205.11916**.

30 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*, 2022, preprint, arXiv:2203.02155, DOI: **10.48550/arXiv.2203.02155**.

31 E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese and C. Xiong, A conversational paradigm for program synthesis, *arXiv*, 2022, preprint, arXiv:2203.13474, DOI: **10.48550/arXiv.2203.13474**.

32 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*, 2021, preprint, arXiv:2107.03374, DOI: **10.48550/arXiv.2107.03374**.

33 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*, 2019, preprint, arXiv:1910.03771, DOI: **10.48550/arXiv.1910.03771**.

34 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*, 2022, preprint, arXiv:2202.01279, DOI: **10.48550/arXiv.2202.01279**.

35 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*, 2022, preprint, arXiv:2201.11903, DOI: **10.48550/arXiv.2201.11903**.

36 A. Fan, M. Lewis and Y. Dauphin, Hierarchical neural story generation, *arXiv*, 2018, preprint, arXiv:1805.04833, DOI: **10.48550/arXiv.1805.04833**.

37 A. Holtzman, J. Buys, L. Du, M. Forbes and Y. Choi, The curious case of neural text degeneration, *arXiv*, 2019, preprint, arXiv:1904.09751, DOI: **10.48550/arXiv.1904.09751**.

38 **https://ur-whitelab.github.io/nlcc-data/**.

39 H. Khlaaf, A hazard analysis framework for code synthesis large language models, *arXiv*, 2022, preprint, arXiv:2207.14157, DOI: **10.48550/arXiv.2207.14157**.

40 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*, 2020, **585**, 357.

41 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, *Comput. Phys. Commun.*, 2010, **181**, 1477.

42 G. Landrum, *et al.*, *Rdkit: A Software Suite for Cheminformatics, Computational Chemistry, and Predictive Modeling*, Greg Landrum, 2013.

43 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 Comput. Biol.*, 2017, **13**, e1005659.
44 C. Nantasenamat, "would be cool to have gpt-3 generate new chemical structures in smiles notation?", Twitter, 1516794237391863810, 2022 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 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 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.
45 C. Nantasenamat, "would be cool to have gpt-3 generate new chemical structures in smiles notation?", Twitter, 1516794237391863810, 2022 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 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 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.
46 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*, 2022, preprint, arXiv:2204.05862, DOI: **10.48550/arXiv.2204.05862**.
47 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.*, 2019, **47**, D1102.
48 C. Edwards, T. Lai, K. Ros, G. Honke and H. Ji, Translation between molecules and natural language, *arXiv*, 2022, preprint, arXiv:2204.11817, DOI: **10.48550/arXiv.2204.11817**.
49 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.*, 2018, **8**, e1340.
50 **https://www.tabnine.com/**.
51 **https://github.com/features/copilot**.
52 **https://openai.com/blog/chatgpt/**.