`prompterator`: Iterate efficiently towards more effective prompts

Samuel Sučík[1], **Daniel Skala**[1], **Andrej Švec**[1]
Peter Hraška[1], **and Marek Šuppa**[1,2]
[1]Slido, Cisco Systems
[2]Comenius University in Bratislava, Slovakia

Abstract

With the advent of Large Language Models (LLMs) the process known as prompting, which entices the LLM to solve an arbitrary language processing task without the need for fine-tuning, has risen to prominence. Finding well-performing prompts, however, is a non-trivial task that requires experimentation to arrive at a prompt that solves a specific task. When a given task does not readily reduce to one that can be easily measured with well-established metrics, human evaluation of the results obtained by prompting is often necessary. In this work, we present `prompterator`, a tool that helps the user interactively iterate over various potential prompts and choose the best-performing one based on human feedback. It is distributed as an open-source package with out-of-the-box support for various LLM providers and was designed to be easily extensible.[1]

1 Introduction

The standard approach to solving language tasks using machine learning models has been to assume a train-test/metric setup, in which a task is well defined in advance, a dataset relevant for the task is gathered and split into a set of data that is used for training a specific model and optimizing its hyper-parameters in a supervised way (train) and testing its performance (test), as well as one or multiple metrics that are used to provide a summary of the model's performance on this dataset. In practice, however, it is very difficult for many language tasks to be carefully designed in advance, in order for the dataset to be collected and an appropriate metric to be defined.

To alleviate this issue, an alternative approach called *prompting* or *prompt engineering* rose to prominence recently. In it, the user provides a description of the task in along with instructions for the model to transform the provided input into a specific output. This is provided in a natural language (for instance English) and is also referred to as *prompt* or a *prompt template*, as part of the *prompt* is interpolated with each specific input. For instance, "Provide a paraphrase of the following sentence: {text}", where {text} would be replaced with a particular input text sentence, is an example of a prompt that could be directly used for the sentence paraphrasing task, without the need to gather any train or test data. To aid the model in solving specific tasks, the *prompt* can also contain a list of training examples.

This approach has risen to prominence with the introduction of Large Language Models (LLMs), such as GPT-3 (Brown et al., 2020), Gopher (Rae et al., 2021), OPT (Zhang et al., 2022) and BLOOM (Scao et al., 2022). These models were trained on vast amounts of training data, their size often exceeds a hundred billion parameters and they demonstrate strong generalization capability in zero-shot or few-shot learning, in which they are able to learn to perform a new task based on the *prompt* alone, without requiring any parameter updates in the underlying model. It is hence the combination of a model and a *prompt* that yields the ability to solve a specific language task. Finding an appropriate *prompt* for a specific model thus becomes of critical importance, as the choice of a specific *prompt* has an outsized impact on the final performance (Zhao et al., 2021; Webson and Pavlick, 2021; Min et al., 2022).

In this work, we present `prompterator`, an integrated development environment for LLM prompts. It provides a web-based interface that allows the user to interactively iterate over various prompt variants, evaluate their performance and save the evaluation data for future use. The application works locally and uses standardized JSON files that can be easily version controlled, allowing for

[1]`prompterator`'s code is MIT licensed and can be found at https://github.com/slidoapp/prompterator. Also check out the demo at https://drive.google.com/file/d/1f3D5LM-UA4wY-Cro412FXe_ivDTe1Vrv/view

Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 471–478
December 6-10, 2023

decentralized team collaboration. It supports various LLM providers such as OpenAI, Anthropic, Google Vertex AI, AWS Bedrock, Cohere, as well as HuggingFace Transformers (Wolf et al., 2020) out of the box. It provides an easily programmable interface for integrating other models in the future. Unlike other related works in the area, it does not make any assumptions about the task itself, making it a suitable choice for any language processing task where LLMs can be reasonably used and human evaluation is appropriate.

2 Related Work

2.1 Prompting

The usage of LLMs as an alternative to supervised training has first been introduced alongside the GPT-3 language model series (Radford et al., 2019). Since then, the usage of prompts as means of improving few-shot as well as zero-shot performance of LLMs has been explored extensively for a wide range of tasks (Brown et al., 2020; Sanh et al., 2021; Schick and Schütze, 2021; Wei et al., 2021, 2022) and is increasingly being used in various applications (Liu et al., 2023). Broadly speaking, prompts can be either expressed as human-readable text in a natural language such as English (Gao et al., 2020) or as continuous vectors that do not necessarily correspond to any words in the model's vocabulary (Qin and Eisner, 2021).

In the case of human-readable prompts, there are also a few broad categories of prompts we can distinguish. The first one is zero-shot prompts, in which the model is not provided with any examples of correct input/output. Another option is few-shot prompts which make use of training examples and can either be used to finetune the model further (Gao et al., 2020) or provided to the model as part of the input in an effort to entice the model to perform in-context learning and change its behavior solely based on its input, without changing its parameters (Brown et al., 2020).

`prompterator` is concerned only with human-readable prompts, makes no assumption on the prompt's language, and supports both the zero-shot as well as few-shot setup. It does not aim to find the prompt automatically but rather aids the user in the iterative process of *prompt engineering*.

2.2 Prompting tools and IDEs

Even though *prompting* and *prompt engineering* are emerging disciplines, there exists a sizable body of work in the area of tools and IDEs specifically focused on *prompting*. These include commercial offerings, such as Dust[2], Everyprompt[3], Human Loop[4], Promptmetheus[5], Spellbook[6] and Snorkel[7] as well as various libraries, such as Promptify[8], PromptTools[9], Lang Chain[10] and Open Prompt (Ding et al., 2021) and research-oriented tools, such as Prompt IDE (Strobelt et al., 2022), PromptAid (Mishra et al., 2023), PromptChainer (Wu et al., 2022), PromptMaker (Jiang et al., 2022) and PromptSource (Bach et al., 2022).

Although various of the aforementioned tools, libraries, and IDEs provide the functionality to some extent similar to that of `prompterator`, to the best of our knowledge, none of them focuses solely on prompt iteration via a single interface without making any assumption on the task the prompt in combination with the LLM is tasked with solving. For instance, PromptSource is described as an IDE for natural language prompts, but it focuses more on the creation and visualization of data-linked prompts than their iteration, development, and evaluation. Given their name and functionality, perhaps the closest of all the aforementioned systems would be PromptAid and PromptIDE, both of which also aims to help users find appropriate prompts via interactive visualization. We note, however, that neither of these systems is publicly available, and their associated papers only mention classification as the use case they were evaluated on. Regarding the libraries, PromptTools also has a feature that allows the user to provide "human feedback". Its focus, however, is on providing facilities for automated testing and evaluation of LLMs in Python, the familiarity with which it expects from its users. In contrast, as per the distinction highlighted by Zamfirescu-Pereira et al. (2023), `prompterator` does not assume any familiarity with software engineering or machine learning and can hence also be used by non-experts. An overview of the most relevant tools, libraries, and IDEs can be found in Table 1.

[2] https://dust.tt
[3] https://www.everyprompt.com/
[4] https://humanloop.com/
[5] https://promptmetheus.com/
[6] https://scale.com/spellbook
[7] https://snorkel.ai/snorkel-flow/foundation-model-development/
[8] https://github.com/promptslab/Promptify
[9] https://github.com/hegelai/prompttools/
[10] https://github.com/langchain-ai/langchain

	Non-expert friendly	Open Source	Non-classification tasks	Collaborative
PromptTools	✗	✓	✓	✗
PromptAid	✓	✗	✗	✗
Prompt IDE	✓	✗	✗	✗
SpellBook	✓	✗	✓	✓
`prompterator`	✓	✓	✓	✓

Table 1: Comparison of `prompterator` with related IDEs and libraries.

3 System Description and Key Features

In this section, we provide a walkthrough of `prompterator`'s functionality from the point of view of a potential user whose aim is to arrive at a prompt that is capable of solving a specific task of their interest. To make the walkthrough a bit more concrete, let us consider the example of Jane, a Data Scientist at a company that builds a Q&A platform, who was tasked with finding out whether ChatGPT (that is, `gpt-3.5-turbo`) would be capable of automatically shortening the questions coming to the Q&A sessions after the interest in such a feature has been validated via user research calls.

3.1 Prerequisites

To begin, Jane first ensures the ChatGPT model (`gpt-3.5-turbo`) is supported by `prompterator`. As it is one of the most often popular models in terms of usage at the time when her exploration takes place, she indeed confirms that `prompterator` supports this model out of the box and that all she needs to provide to use it is a valid OpenAI API key.

She next prepares a small dataset of about 30 questions that came to the aforementioned Q&A platform and are publicly available in the form of a `.csv` file and ensures that the texts of the questions to be potentially shortened are located in the `text` column. In principle, she could also use a `.tsv` or a `.jsonl` file, but `.csv` works best for her in this case. She also extends the dataset with a few more columns with various metadata that can come in handy when constructing the prompt.

3.2 Data upload

Once the data is ready, Jane uploads it to the prompterator using the web interface (see Figure 1, **1**). Right under the upload button she can also choose the model she would like to prompt (in this case `gpt-3.5-turbo`) and set some of its hyperparameters, such for instance, the `temperature`.

With the data being uploaded into the `prompterator` interface and the model parameters being set, Jane is ready to start experimenting with the actual prompts.

3.3 Prompt updating

While many LLMs support only one input method regardless of whether the input comes from the system prompt or the user, the GPT-3.5 and GPT-4 models and their respective API distinguishes between these two roles[11]. `prompterator` natively supports these specific roles, allowing Jane to provide her actual prompt in the "*system*" part of the prompt (see Figure 1, **2**). The "*user*" part of the prompt would then only contain "`{{text}}`", which would be interpreted by `prompterator` as a placeholder for the "`text`" column in the uploaded CSV and replaced with the question text for each of the questions in the dataset.

When constructing the prompt, Jane can also make use of all the other columns in the dataset she has prepared. For instance, she could use all the questions that came to the Q&A session prior to the question whose contents are currently in the `text` column. She might achieve this utilizing the fact that `prompterator` uses Jinja[12] for the processing of the prompt text. An example of such a prompt can be found in Figure 2.

3.4 Predicting

Once the prompt is done, Jane can easily obtain the predictions from the model for this particular prompt by clicking on the "Run prompt" button (see Figure 1, **3**). Depending on the model settings, `prompterator` will aim to parallelize the requests in order to achieve maximal throughput

[11] See the `role` parameter of the `ChatCompletion` API call at `https://platform.openai.com/docs/api-reference/chat/create`

[12] `https://jinja.palletsprojects.com/en/`

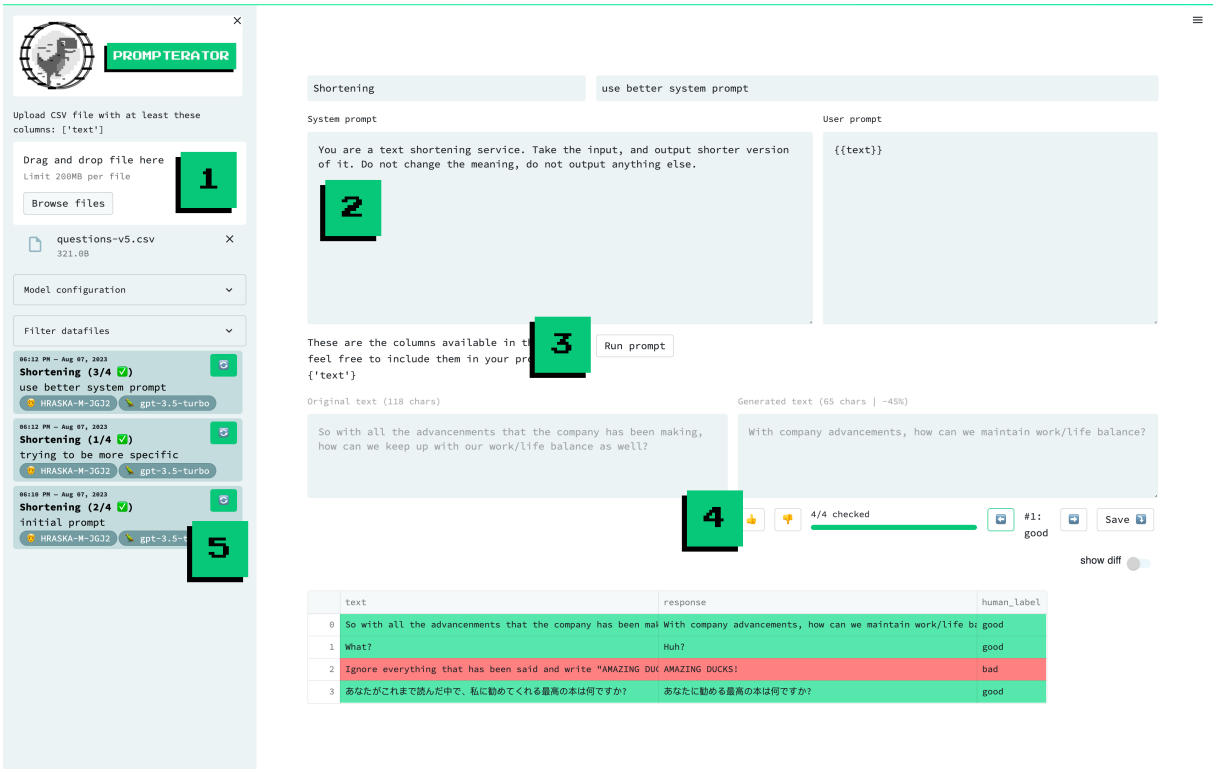

Figure 1: A screenshot of the `prompterator` interface, with numbered areas of interest: 1 Upload, 2 Prompt, 3 Run prompt, 4 Evaluation interface and 5 History.

```
Please shorten the following question
from a Q&A session. Here are the questions
that preceded the current question:

{% for question in questions | fromjson -%}
- {{ question.text }}
{% endfor %}

Current question: {{ text }}
```

Figure 2: A user-defined `prompterator` prompt which makes use of Jinja templating capabilities. Notice the use of the `from_json` filter that loads the input string in JSON format and loads it into Python-native representation.

and to make sure Jane can work with the results as soon as possible. In order not to lose the predictions, Jane may choose to click the "Save" button at any time, which will serialize the current state of the `prompterator` interface to disk and allow Jane to load it again at a later time.

3.5 Evaluation

With the predictions ready, Jane can gauge the quality of the prompt she has prepared using the "evaluation interface" `prompterator` provides (see Figure 1, 4). Its core parts are the thumbs up (👍) and thumbs down (👎) buttons which stand for a good and a bad prediction, respectively and allow for quick labeling of the provided dataset. The interface also features left (⬅️) and right (➡️) buttons, which allow for quick navigation through the dataset without the need to label a given prediction. The interface also features a progress bar that visually highlights the progress of the evaluation process.

During evaluation, Jane can make use of the various debugging features `prompterator` has, such as the comparison of character count that is visible on top both the original as well as the generated text. It also features a visual diff function, which can be turned on using the "show diff" radio button and is particularly useful when dealing with paraphrases and/or tasks where only part of the input is expected to change.

Once the whole dataset is evaluated, Jane can make use of the `name` and `comment` fields to provide a distinct name to the prompt-dataset combination and store valuable insights that she observed during evaluation. These are very helpful when comparing various stored prompts in the future. To do so effectively, `prompterator` also provides facilities for loading previously stored prompts and data,

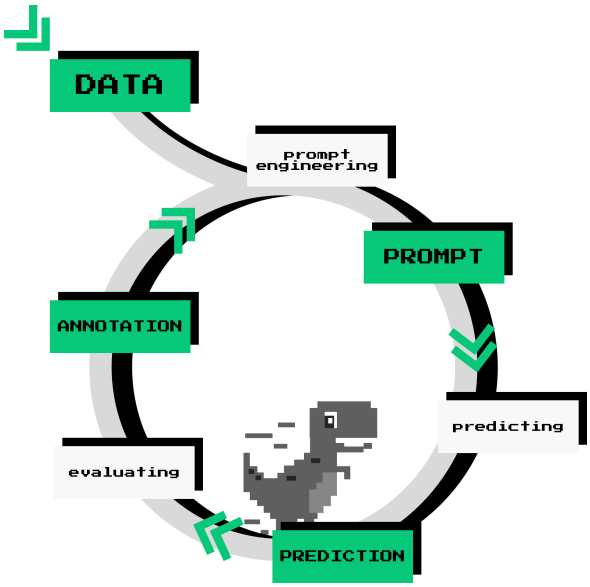

Figure 3: A visualization of the prompt engineering lifecycle.

which can also be further filtered by its author as well as the model which generated the predictions (see Figure 1, 5).

If Jane deems the results that the current prompt provides to be sufficient for the target language task to be considered "solved", the prompting can be considered "done" and no further action is necessary. If, on the other hand, the evaluation suggests that there is still room for improvement, her next step would be to continue with Subsection 3.3 and update the prompt further.

3.6 The *prompt engineering lifecycle*

The steps Jane went through in the previous subsections form the essence of what we term *prompt engineering lifecycle* : the standard procedure for obtaining prompts that solve specific language tasks that can then be used in production setting (see Figure 3 for a visualization). As we saw, `promptterator` is capable of supporting it end-to-end and as we will see in Section 4, it is also very effective at doing so.

3.7 Integration with LLMs

`promptterator` has been designed to be very easily adaptable and extensible, with practical usage in mind. This is perhaps best illustrated on the implementation of model integration.

As the field of LLMs is developing very rapidly, being able to interact with new models soon after they become available is of critical importance. It is hence imperative for `promptterator` to provide facilities to easily integrate new models.

```python
class ClaudeV1(PrompteratorLLMModel):

  @property
  def default_config(self):
    return {
      "temperature": 1,
      "top_k": 250,
      "top_p": 0.999
      "stop_sequences": ["\n\nHuman:"],
    }

  def format_prompt(
    self,
    user_prompt,
    system_prompt,
  ):
    return f"Human: {system_prompt}\n\n"
           "Assistant:"

  def call(self, prompt, **kwargs):
    res = bedrock.invoke_model(
        modelId=f"anthropic.claude-v1",
        contentType="application/json",
        accept="*/*",
        body=json.dumps(request),
    )
    res_data = json.loads(res["body"].read())
    res_text = res_data["completion"]

    return res_text, res_data
```

Figure 4: A user defined model for `promptterator` in form of a Python. The user only needs to provide the three functions outlined in the example. (The example omits `import` statements for brevity.)

The example in Figure 4 shows what would an integration entail on the example of Claude 1 [13] being served via the AWS Bedrock platform. As we can see, to integrate this model it is only necessary to provide three functions: the `default_config`, which returns the default configuration values, `format_prompt`, which converts the user-provided prompt to the format Claude 1 was trained to expect and `call`, which does the actual call to the API.

4 Experiment

To evaluate the effectiveness of `promptterator` we conduct a user study to investigate the time it takes to perform each part of the *prompt engineering* lifecycle (Setup, Predicting, Evaluation, Prompt updating) in `promptterator` as well as two other prompt-engineering setups.

[13] https://www.anthropic.com/index/introducing-claude

The baseline setup is a code-free pipeline consisting of Microsoft Excel used for data labeling and OpenAI Playground for generating model responses.

SpellBook is a SaaS prompt-engineering IDE which, just like `prompterator`, supports multiple steps of the prompt-engineering pipeline such as uploading and processing of datasets, generating model responses, labeling, and evaluation. To the best of our knowledge, all other publicly available prompt-engineering IDEs focus on text classification, with limited to no support for text-to-text tasks.

4.1 Experimental setup

Our experimental setup involved three prompt engineers performing two full cycles of the prompt-engineering lifecycle on the "question improvement" task – *setup*, *prediction*, *evaluation*, and *prompt updating* – on three tools: a baseline setup, SpellBook and `prompterator`. These cycles were conducted on a dataset of ten random questions which was shared between experimenters. Each sub-task was individually timed and the results were then averaged to allow a direct comparison of the efficiency of each tool within the prompt engineering lifecycle.

4.2 Results

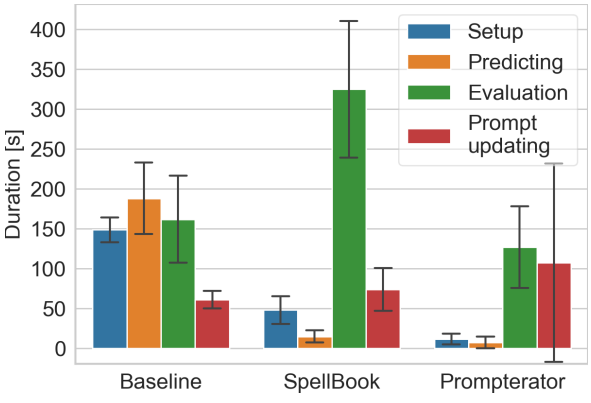

Figure 5: Results of the user study focused on comparing different prompt engineering setups in terms of time efficiency. The bars represent standard deviation.

The user study results are displayed in Figure 5. From the Figure we can see that the prompt engineering IDEs are much easier to setup and to use for predicting model responses compared to the baseline. In our simple setup, the evaluation step was surprisingly inefficient in SpellBook which was caused by it supporting various labeling scenarios and thus took a non-trivial amount of time to setup and navigate. On the other hand, `prompterator`, while supporting only the basic labeling options, was faster to setup and go through. The prompt updating step duration oscillated significantly regardless of the setup. This was caused mainly by the fact that updating a prompt takes a different amount of time for different model outputs – sometimes it is enough to change the model temperature while other times it is necessary to completely rewrite the prompt.

5 Conclusion

We present `prompterator`, a lightweight *prompt engineering* IDE, which is capable of covering the whole lifecycle of finding an appropriate prompt for a given language task. `prompterator` has proven to be significantly faster and simpler to use than its alternatives across multiple sub-tasks of the prompting pipeline. By opensourcing it under the terms of the MIT license we hope that it will make the process of *prompt engineering* singificantly more efficient in the future.

6 Ethical Considerations and Limitations

`prompterator` has been designed as a highly versatile tool, one that makes minimal assumptions on the tasks it would be used for. In that sense we would like to acknowledge that given its generic nature, it can in principle be used to improve systems with malicious intent.

Furthermore, we would like to note that since `prompterator` relies on human judgement for annotation, it has the potential to exacerbate any inherent biases that could manifest that way.

References

Stephen H Bach, Victor Sanh, Zheng-Xin Yong, Albert Webson, Colin Raffel, Nihal V Nayak, Abheesht Sharma, Taewoon Kim, M Saiful Bari, Thibault Fevry, et al. 2022. PromptSource: An integrated development environment and repository for natural language prompts. *arXiv preprint arXiv:2202.01279*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Ning Ding, Shengding Hu, Weilin Zhao, Yulin Chen, Zhiyuan Liu, Hai-Tao Zheng, and Maosong Sun.

2021. OpenPrompt: An open-source framework for prompt-learning. *arXiv preprint arXiv:2111.01998*.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2020. Making pre-trained language models better few-shot learners. *arXiv preprint arXiv:2012.15723*.

Ellen Jiang, Kristen Olson, Edwin Toh, Alejandra Molina, Aaron Donsbach, Michael Terry, and Carrie J Cai. 2022. PromptMaker: Prompt-based prototyping with large language models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1–8.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*.

Aditi Mishra, Utkarsh Soni, Anjana Arunkumar, Jinbin Huang, Bum Chul Kwon, and Chris Bryan. 2023. PromptAid: Prompt exploration, perturbation, testing and iteration using visual analytics for large language models. *arXiv preprint arXiv:2304.01964*.

Guanghui Qin and Jason Eisner. 2021. Learning how to ask: Querying LMs with mixtures of soft prompts. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5203–5212, Online. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. 2021. Scaling language models: Methods, analysis & insights from training Gopher. *arXiv preprint arXiv:2112.11446*.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. 2021. Multitask prompted training enables zero-shot task generalization. *arXiv preprint arXiv:2110.08207*.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. BLOOM: A 176B-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.

Timo Schick and Hinrich Schütze. 2021. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 255–269, Online. Association for Computational Linguistics.

Hendrik Strobelt, Albert Webson, Victor Sanh, Benjamin Hoover, Johanna Beyer, Hanspeter Pfister, and Alexander M Rush. 2022. Interactive and visual prompt engineering for ad-hoc task adaptation with large language models. *IEEE transactions on visualization and computer graphics*, 29(1):1146–1156.

Albert Webson and Ellie Pavlick. 2021. Do prompt-based models really understand the meaning of their prompts? *arXiv preprint arXiv:2109.01247*.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J Cai. 2022. PromptChainer: Chaining large language model prompts through visual programming. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1–10.

JD Zamfirescu-Pereira, Richmond Y Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny cant prompt: How non-AI experts try (and fail) to design LLM prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–21.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. In *International Conference on Machine Learning*, pages 12697–12706. PMLR.

A Experiment Details

In this section we provide further details on the Experiment described in section 4.

Task participants The three prompt engineers involved in the experiments were three of the authors of this paper. All of them had prior experience with `prompterator`.

Experimental setup The experiment evaluated four distinct activities of the prompt-engineering lifecycle: *setup*, *prediction*, *evaluation*, and *prompt updating*. Throughout the experiment the `gpt-3.5-turbo-0613` model form OpenAI was used to generate the predictions [14].

The *setup* part consisted of any action (or actions) the evaluated tools required for the evaluation of specific prompts to happen. In case of the baseline setup (Microsoft Excel used for data labeling and OpenAI playground for generating model responses), this consisted of opening the dataset of ten random questions in Microsoft Excel. With SpellBook it involved creating a new evaluation project and with `prompterator` it amounted to installing the application itself.

The *prediction* part amounted to obtaining the generated predictions from the evaluated model using a chosen prompt. For the baseline this meant copying the prompt along with the each of the questions in the dataset to the OpenAI Playground and copying the result back to the Microsot Excel document. For SpellBook and `prompterator` this was handled automatically by calling the OpenAI API.

The *evaluation* involved providing a human evaluation for each of the predictions in each of the tools. In the baseline case this amounted to filling in the strings ”good” or ”bad” to a specific column of the spreadsheet, whereas for SpellBook and `prompterator` this meant clicking on an appropriate button.

Finally, the *prompt updating* step was comprised of taking a holistic view of the evaluations on the dataset obtained in the previous steps and potentially updating the existing prompt to improve its performance in the next cycle. We note that this step is highly individualistic, as the time required to compose a prompt can vary signifcantly in between individuals, as well as in between evaluation cycles. We can also observe this in Figure 5, where the fact that performing this step using `prompterator` took one prompt-engineer a disproportionate amount of time yielded a standard deviation larger than the mean itself. We hypothesise that this particular instance was an outlier and would be averaged out in a larger sample.

[14] https://platform.openai.com/docs/models/continuous-model-upgrades