Model-Driven Prompt Engineering

Anonymous author(s)
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—Generative artificial intelligence (AI) systems are capable of synthesizing complex content such as text, source code or images according to the instructions described in a natural language prompt. The quality of the output depends on crafting a suitable prompt. This has given rise to *prompt engineering*, the process of designing natural language prompts to best take advantage of the capabilities of generative AI systems.**

Through experimentation, the creative and research communities have created guidelines and strategies for creating good prompts. However, even for the same task, these best practices vary depending on the particular system receiving the prompt. Moreover, some systems offer additional features using a custom platform-specific syntax, *e.g.*, assigning a degree of relevance to specific concepts within the prompt.

In this paper, we propose applying model-driven engineering to support the prompt engineering process. Using a domain-specific language (DSL), we define platform-independent prompts that can later be adapted to provide good quality outputs in a target AI system. The DSL also facilitates managing prompts by providing mechanisms for prompt versioning and prompt chaining. Tool support is available thanks to a Langium-based Visual Studio Code plugin.

***Index Terms*—prompt engineering, model-driven engineering, domain-specific language, generative AI, large language models**

I. Introduction

Generative artificial intelligence (AI) is currently attracting a lot of attention as a potentially disruptive technology [1]–[3]. Generative AI aims to automatically construct complex artifacts that comply with a set of input requirements. The format of the produced output varies: natural language text (ChatGPT, Bard, Bing Chat, Claude), source code snippets (Copilot, Ghostwriter, Tabnine), music (MuseNet, MusicLM), images (DALL-E, Midjourney, Stable Diffusion), ... Regarding the input, many generative AI systems are based on a *prompt*, a natural language description of the goals of the generated artifact [4], [5]. More recent systems also accept *multi-modal* inputs, *e.g.*, an image in addition to a prompt. In these systems, the extra inputs may provide additional information or be used as seeds or examples to guide the generation process. Nevertheless, the textual prompt continues to be the core input mechanism.

Several reasons motivate the growing interest in generative AI. First, the availability of large datasets and computing power to train generative systems enables them to be increasingly more effective. Furthermore, using natural language as the input format improves usability and is flexible enough to support numerous application scenarios.

Nevertheless, a shortcoming of generative AI systems is that the quality of the output depends on the structure of the input prompt [6]–[12]. As a result, the AI community has proposed a set of strategies, best practices, guidelines and supporting tools for creating good prompts. The process that involves crafting, evaluating and fine-tuning prompts is called *prompt engineering*.

A challenge of prompt engineering is the fact that it is essentially a *platform-specific* process. Prompts that perform well in a particular system may under-perform in other systems [13], [14], or even in different versions of the same system. For instance, the release of version 2 of the Stable Diffusion open-source text-to-image system caused user complaints that prompts previously considered as "good" were producing worse results[1]. Two changes motivating the performance gap were different training datasets and the replacement of a component of the AI system (the encoder). These types of changes are expected each time a new version is released.

Moreover, some advanced aspects of prompting such as negative information, assigning weights to prompt fragments or setting platform-specific flags have a different degree of support and use a different syntax in each generative AI system.

To face these challenges, we propose to apply model-driven engineering (MDE) principles and techniques to prompt engineering. Our goals are: (a) facilitating the definition of good prompts for different generative AI systems; (b) facilitating the migration of a prompt from one platform to another without losing effectiveness; and (c) enabling knowledge management of prompt-related tasks (documentation, traceability, versioning, ...). In particular, we offer the following contributions:

- We propose Impromptu, a domain-specific language (DSL) for defining prompts in a platform-independent way. The DSL supports modular prompt definition and is able to describe different tasks (text-to-text, text-to-image, ...) as well as multi-modal inputs.
- We use code generation to produce fine-tuned versions of the prompt for specific platforms. We provide an example for two text-to-image generative AI systems: Midjourney and Stable Diffusion.

Funding agency information removed for anonymous submission.

[1]https://thealgorithmicbridge.substack.com/p/stable-diffusion-2-is-not-what-users

- We characterize further potential contributions and long-term vision of MDE in the context of prompt engineering.

Naturally, this DSL and toolkit are not intended for prompts that will be used just once and then thrown away, *e.g.* in a trial-and-error manner. Instead, the contributions of this paper are intended for recurring prompts, *e.g.*, prompts that are valuable for a particular organization because they are used as part of a business process.

Paper organization: The remainder of the paper is organized as follows. Section II discusses related work on prompt engineering and tool support. Then, Section III presents our DSL for prompt engineering and the toolkit that implements it. Section IV discusses the generation of platform-specific prompts. Then, Section V analyzes how MDE can contribute to the field of prompt engineering. Finally, Section VI concludes.

II. Related work

The massive impact of the input prompt on the output of a generative AI system has triggered the creation of a myriad of prompting tools, guides and strategies to help craft the right prompt for the task at hand. In this Section, we summarize these different initiatives and compare them with our proposal.

A. General prompting strategies

Prompts can be used to ask generative AI systems to solve tasks even when we provide few examples (*few-shot*) or even none (*zero-shot*) [4], [8], [15]. Given a task, there are several strategies to craft a suitable prompt:

- **Experimentation:** Starting from an initial prompt, iteratively refine it after inspecting the output artifact. The goal is to craft a prompt that produces outputs that are more suitable for our problem. As a community effort, generative AI users have created several *guidelines* for creating good prompts for typical tasks.
- **Preset/Recipe library**: Reuse a predefined prompt template for a particular task which has been previously defined in a library, repository or collection of prompts.
- **Reverse engineering:** Infer suitable prompt contents from a target example.
- **Prompt optimization:** Given an initial prompt, feed it to another generative AI model that automatically suggests edits that are likely to improve the quality of its output. It is also possible to do this optimization manually by applying known heuristics and best practices. For example, a reasoning task may achieve a better result if we request the list of intermediate arguments and deductions (a *chain-of-thought* [9]).
- **Batch prompting [16]:** Group several queries into a single prompt to reduce the costs (computational, environmental, and economic) of invoking a generative AI system.
- **Prompt chaining [17]:** Decompose a prompt for a complex task into a sequence of intermediate subtasks, creating a prompt to perform each step separately to achieve a greater precision.

Table I describes, for each of the key strategies above, the expected input, the number of interactions with the generative AI system and the human role in the process. Note that even though in some strategies the prompt engineer receives assistance, his/her role is still critical in the proposal and validation of prompts. In the following Sections, we discuss the tools support available to prompt engineers.

B. Specific prompting strategies and tools

Most current approaches to create or manage prompts are specialized in a given task or, a type of output artifact.

We have, for instance, prompting tools focusing on text-to-text generation. Among them, we would like to highlight PromptIDE [18], PromptArray[2], PromptSource [19], Promptable[3], PromptMaker [20] or Promptify [4]. Additional discussion on prompts for natural language tasks is provided in [5].

Others are more oriented to text-to-image generation, such as PromptBuilder[5], PromptGen [6], PromptMakr[7], PromptMaker[8], DiffusionBee [9], even covering the generation of 3D CAD images [21]. A more complete discussion of prompts for image generation is provided in [22] and [23].

While each of these tools offers its own prompting interface, most are rather simple and basically consist of a free writing area to indicate the prompt and a visualization of the prompt execution. A few also execute prompts on the generative AI system and log the results for visualization or analysis. Nevertheless, there is no advanced support for the writing of the prompt itself via some type of prompt language or similar, nor a way to document and version a prompt. We discuss a few exceptions in the next subsection.

C. Prompt builders

Some tools aim to *assist* the prompt engineer when writing the prompts, mostly to offer a taxonomy of a fixed set of prompt dimensions you can choose from when building your prompt. Examples are PromptGen[10], Eye for AI [11] or PromptSource[12] while works such as [24] or [25] are more focused on describing possible taxonomies and catalogs to be integrated in future prompt builders. PromptIDE [18] goes one step further and allows users to experiment with prompt variations and iteratively optimize prompts. Note that, each of these tools focuses on a specific type of output and often even on a specific generative AI system.

Other tools focus on the connection of prompt dimensions (*e.g.*, as you can do in PromptArray[13] offering a reduced set

[2]https://github.com/jeffbinder/promptarray
[3]https://promptable.ai/
[4]https://github.com/promptslab/Promptify
[5]https://aitextpromptgenerator.com/builder
[6]https://promptgen.vercel.app/
[7]https://promptmakr.com/
[8]https://promptmaker.com/
[9]https://diffusionbee.com/
[10]https://promptgen.vercel.app/
[11]https://eyeforai.xyz/
[12]https://github.com/bigscience-workshop/promptsource
[13]https://github.com/jeffbinder/promptarray

TABLE I
A CATALOG OF PROMPTING STRATEGIES.

Strategy	Input(s)	Number of calls to AI system	Human role
Iterative refinement	Prompt sketch	Several (one per iteration)	Full control
Preset library	Prompt template + instantiation	One	Create templates + Search suitable template
Reverse engineering	An example	One (+ call to validator)	Identify example + Validate suggested prompt
Prompt optimization	Prompt sketch	One (+ call to optimizer)	Propose sketch + Validate optimizations
Batch prompting	Set of prompts	One	(Optionally) Propose initial prompts
Prompt chaining	Prompt sketch	Several (one per subtask)	Full control

of boolean operators to mix partial prompts) or the chaining of prompts (*e.g.*, PromptChainer [26], with a visual language for chains, or LangChain[14], a Python library, focus on text-to-text prompts). The LlamaIndex library [27] also helps users connect prompts with external data sources that can be referenced as part of the prompt pipeline. Similarly, [28] mixes scripting with prompt descriptions for a more "programming-like" experience. As before, these tools are available for a limited set of languages and AI systems.

Finally, PromptPerfect[15] and Promptist[16] explore the idea of optimizing user prompts for specific platforms though this optimization is automatic and therefore opaque to the prompt designer. The automatic generation of prompts is also subject of investigation [11], [29] so far with limited results.

Impromptu aims to gather the best ideas of these approaches to provide the first platform-independent prompt DSL together with the infrastructure to assist in the creation of prompts and their versioning and automatic transformation to specific concrete target platforms providing, this way, an integrated end-to-end platform for prompt engineering.

III. A DSL FOR PROMPT ENGINEERING

In this Section, we motivate the need for a prompt DSL and present the Impromptu DSL for describing prompts and prompt chains in a platform-independent way.

A. Why a DSL?

First, we discuss the suitability of creating a DSL for prompt engineering. Generative AI systems operate on natural language, a formalism which is sufficiently expressive and flexible to describe any type of task or scenario. Then, why is it necessary to use a DSL when it is possible to use natural language? Several reasons motivate this decision:

Prompt templates: For the sake of generality, it may be desirable to define a prompt as a *template*, where fragments of the prompt are parameters to be instantiated in each particular call with a different piece of information or command. For instance, in a question answering prompt, we may have the input question and the desired length of the response as parameters. Thus, the definition of the prompt should allow defining such parameters.

Hyperparameters: The call to a generative AI system requires setting suitable values for hyperparameters, like the *temperature*, which controls the randomness of the output (random outputs may exhibit more errors but also more creativity). It is useful to record information about suitable values for these hyperparameters together with the prompt.

Multi-modal inputs: Recent generative AI systems [30]–[32] support additional inputs beyond textual prompts, such as images. It is necessary to describe these additional inputs and their role in the definition of the task.

Modularity: A company may desire that several prompts reuse a set of instructions to specify the tone, mood or target audience of the output, specify fallback actions (*e.g.*, provide a help message), or specify certain limitations on the output (*e.g.*, forbidden actions or topics). Rather than manually replicating and appending these snippets each time, it would be desirable to define these snippets independently and import them where it is necessary.

Chaining: Complex tasks may not be solvable by means of a single prompt. In this scenario, it is necessary to indicate the relationship between different prompts in a chain, *e.g.*, when a given prompt uses the result from a previous one.

Documentation and meta-data: A complex prompt is the result of a careful design and extensive experimentation. Nevertheless, this knowledge is lost when we view a prompt simply a string of text. It is necessary to keep track of relevant meta-data, such as its author, versions, and comments.

Cost-effectiveness: Generative AI tools are efficient enough to enable *iterative refinement*: a user proposes a prompt and, if the outcome is not adequate according to some criteria, the user modifies the previous prompt to address the identified shortcoming. The process continues until a satisfactory output is achieved. In this context, intermediate prompts will be thrown away prompts. Similarly, many prompts may be one-offs, that will never be reused. In these scenarios, there is limited benefit in putting the effort to describe them using a DSL. Instead, Impromptu is targeted towards complex prompts that aim to be re-used. We are considering the prompts that empower an intelligent application or service relying on a generative AI system as its back-end. These are the types of prompts where it makes sense to optimize the effectiveness, minimize cost and enable platform-independence.

Flexibility: Natural language is capable of describing any type of task or scenario. Instead, a DSL introduces a structure which may limit the freedom provided by natural language. However, a DSL is better suited to expressing *concepts* and

[14]https://langchain.readthedocs.io/en/latest/index.html
[15]https://promptperfect.jina.ai/
[16]https://huggingface.co/spaces/microsoft/Promptist

adapting them to different platforms. For example, thanks to an MDE framework like Impromptu, it is possible to specify that we want to generate an image with the highest possible quality and then have transformations that generate the most suitable quality term for each text-to-image system.

B. Abstract syntax

Figure 1 presents the core concepts of the Impromptu meta-model.

Assets: The central notion being described is the *asset*, which can be either *prompt*, a *composer* or a *prompt chain*. Assets can be annotated with useful information such as language, description and authors. To keep track of different versions of an asset, it is possible to provide a version name, date of creation and last update and identify prior versions of the same prompt. Moreover, assets can be given *labels* to facilitate organizing and searching catalogs of prompts.

Each asset has as a single output that can be of a different type: text, image, ... On the other hand, assets may have several inputs, which can either be textual *parameters* (parts of a prompt whose value is provided by the user or computed by a previous prompt in a chain) or *multi-modal inputs* (such as an image) that provide additional information regarding the input prompt.

Finally, it is possible to specify suitable values for the hyperparameters of AI systems that should be used when generating an output for the asset.

Prompts: Prompts are defined as a sequence of *snippets*. One snippet is identified as the *prefix* (suffix) and is prepended (appended) to the rest of the prompt, while the rest form the core of the prompt.

A snippet can contain a textual literal, a reference to one of the prompt's parameters or to another asset to be reused (allowing for modular definitions of prompts). Moreover, a snippet can also include a property about the output (called *trait*) chosen from a predefined catalog. Figure 2 presents the traits that are independent of the format of the prompt's output. Some sample traits are: including certain elements, emulating the style of a given author, satisfying specific size constraints, ... There are also format-dependent traits, such as the language register (formal, vulgar, ...) for textual outputs or the art form (photography, drawing, ...) for image outputs.

Each component of a prompt, such as snippets or inputs, can be given a *weight* that indicates their relevance.

Composers: A composer is a utility that concatenates several snippets without performing any additional processing. This is useful to collate the output of different text-to-text prompts into a single string.

Prompt chains: A prompt chain is a sequence of prompts that computes a result step by step. Each step corresponds to the execution of an asset. Just like a prompt, a chain has inputs that can be parameters or non-textual data. The inputs of each step of the chain are either set to a constant value, a parameter of the chain or the output of a preceding step. Finally, the last step in the sequence computes the *result*, *i.e.*, the output of the chain.

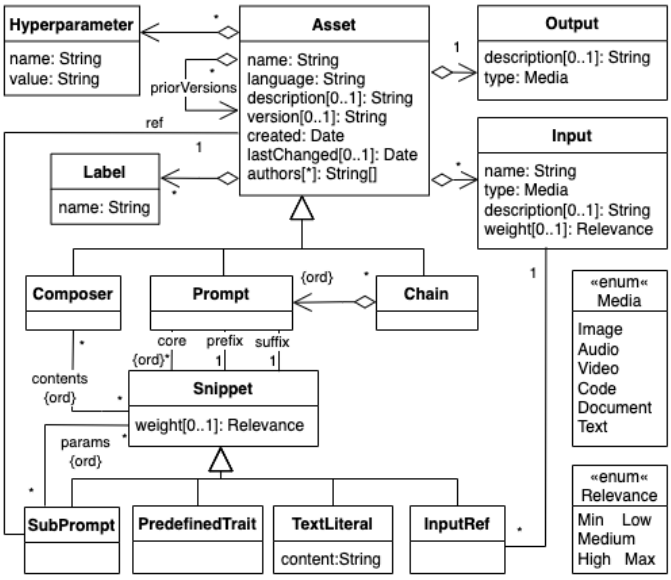

Fig. 1. Core of the Impromptu meta-model.

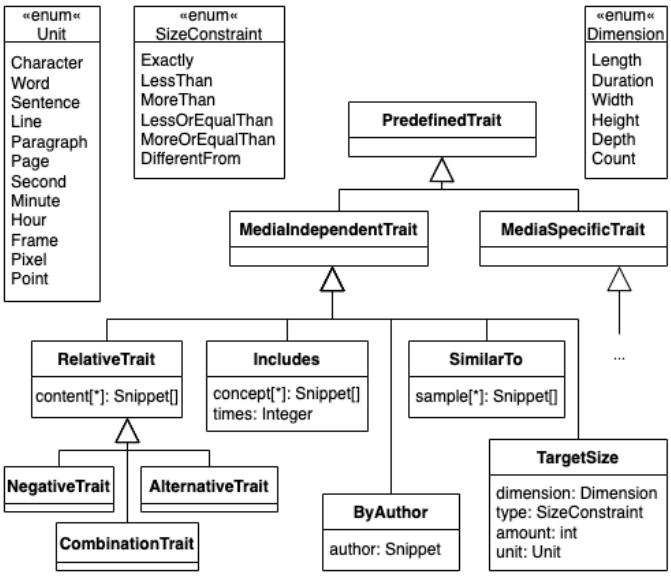

Fig. 2. Format-independent predefined traits.

C. Concrete syntax

We have defined a textual syntax to encode specifications written in the Impromptu DSL. We will present this syntax using an example: the creation of a prompt chain to generate the exercises in an English test.

The exam will consist of three blocks: a set of grammar questions, an AI-generated image and a set of questions about the image. These blocks, the image and all the questions, will be generated by different AI services. The chain will have two parameters: the topic of the image to be generated and the level of competency in English to be assessed.

First, we will define a prompt to be reused as a suffix for several prompts. This prompt provides instructions regarding

the tone and difficulty of the questions. It includes a single parameter: the level of competency in English of the questions.

```
composer Exam.DetailedInstructions
  (@level "English level")

  "Questions should be clear and avoid
  controversial topics such as
  politics or religion.",
  "The difficulty should be adequate
  for students with an English level of ",
  @level, "."
```

We can then provide a prompt to generate the grammar questions:

```
prompt Exam.GrammarQuestion(): text

  core = "Propose 5 grammar questions
         for an English exam",
         "Use British English at all times"
           weight high
  suffix = Exam.DetailedInstructions(@level)
```

The following prompts generate an image about a given topic and a set of questions about that image:

```
prompt GenerateImage(
  @animal "Name of an animal"): image

  core   = "Image of ",
           @animal,
           " in its natural habitat"
  traits = quality(high),
           art_form(photography)

prompt ImageQuestions(
  $image  "Image",
  @animal "Animal appearing in the image",
  @level  "English level)": text

  core = "Propose 5 questions for an English
          course regarding the picture of",
          @animal
  suffix = Exam.DetailedInstructions(@level)

composer CompleteExam(
  @grammar "Grammar questions",
  @questions "Questions about the image")

  "Answer the following questions:\n",
  @grammar, "(50%)\n",
  @level, "(50%)\n"
```

Finally we define the complete chain as follows:

```
chain GenerateExam(
  @level  "English level",
  @animal "Animal in the image"): text

  @q1 = GrammarQuestions(@level)
  $img = GenerateImage(@animal)
  @q2 = ImageQuestions($img, @animal, @level)
  result = CompleteExam(@q1,@q2)
```

D. Implementation

As a prototype, we have implemented an editor for the Impromptu DSL as a Visual Studio Code extension[17]. The implementation is written in TypeScript and is based on Langium [33], an open-source language engineering tool which supports the Language Server Protocol. Thanks to Langium, the editor offers built-in syntax-highlighting, syntax checking and autocomplete suggestions. These suggestions are a useful way to help a user create a prompt, *e.g.*, by suggesting a list of candidate predefined traits.

In addition to a graphical user interface (GUI), Impromptu also offers a simple command-line interface (CLI) that makes it possible to generate tool-specific prompts within automated workflows.

IV. Prompt customization

Platform-independent prompts facilitate *prompt migration*: changing the generative AI system that is consuming prompts. In order to illustrate the capabilities of the Impromptu framework, we have implemented generators that transform the platform-independent prompt into the specific prompt for two particular text-to-image generative AI systems: Midjourney[18] and Stable Diffusion[19] (in the following, SD).

First, we highlight the following differences between the syntax of prompts in these two systems:

- **Attention:** SD can increase the attention devoted to specific concepts using `(concept)` to increase attention or `[concept]` to decrease it. These modifiers can be stacked multiplicatively, *e.g.*, `((concept))` to further increase the weight of a concept. Alternatively, it can assign a weight explicitly using `(concept:weight)`, where weight is a real. Meanwhile, Midjourney assigns weights to individual concepts using the syntax `concept::weight`, where weight is an integer (Midjourney versions 1-3) or real (Midjourney version 4).
- **Negative information**: SD specifies concepts that should be avoided in the image as a separate *negative prompt*. Meanwhile, Midjourney embeds negative information in the same prompt, either using a negative weight or using the `--no` flag preceding a concept.
- **Combinations:** SD allows describing an intermediate concept between two other concepts (*keyword blending*) using the syntax `[concept1:concept2 :N]`, where `N` is a value between 0 and 1 (the larger it is, the more weight is given to the second concept). Midjourney does not offer such feature. Instead, Midjourney's syntax `{concept1, ..., conceptN}` has a different semantics, a *permutation prompt*: it generates one separate prompt per alternative. Thus, combinations need to be described explicitly using a textual description rather than a custom keyword.

[17]The Impromptu VS Code extension is available at the following URL: https://github.com/undisclosed-author/Impromptu/.

[18]https://www.midjourney.com/

[19]In particular, we are targeting AUTOMATIC1111 webui for Stable Diffusion (https://github.com/AUTOMATIC1111/stable-diffusion-webui).

```
composer GenerateGryphoon(): image
  Mixture("eagle", "lion")

prompt Mixture(@animal1, @animal2): image
  core = audience( "children" ) weight high,
         medium( drawing ),
         between(@animal1, @animal2),
         no("violent" ), no("scary")
```

(a)

```
Positive prompt:
  drawing of [eagle:lion :0.5], (for children)
Negative prompt:
  violent, scary
```

(b)

```
drawing of a combination of eagle and lion,
for children::2, --no violent, --no scary
```

(c)

Fig. 3. Example of platform customization: (a) Impromptu platform-independent prompt, (b) Generated prompt for Stable Diffusion, (c) Generated prompt for Midjourney.

Then, Figure 3 shows the process for building a prompt that generates an illustration for a children's book for each system thanks to Impromptu. The illustration should include a fantastic animal combining features from two animals. Fig. 3(a) shows a potential Impromptu prompt, from which our tool can generate platform-specific prompts for SD (Fig. 3(b)) and Midjourney (Fig. 3(c)). Notice that, in the case of SD, the generated prompt has two separate components: the positive prompt and the negative one, with concepts to be avoided.

V. Contributions of MDE to prompt engineering

The previous Sections have illustrated the use of a DSL to support modular prompt definition, prompt migration and prompt documentation. Beyond these features, the use of an MDE-based prompt engineering platform like ours can contribute to several aspects of the prompt engineering process.

Platform selection: When prompts are specified in a platform-independent way, it is possible to implement strategies for selecting the most suitable generative AI system for a given prompt automatically. The criteria used in this decision may include:

- Cost of the AI service.
- Length of the prompt (for services with a prompt limit).
- Support for constructs used in the prompt (*e.g.*, Midjourney does not currently support combination prompts).
- Performance information of different generative AI systems for a particular type of task.
- Available interfaces of the AI service (*e.g.*, API, graphical user interface, ...).

Code generation and run-time management: In addition to generating textual prompts, it is possible to generate the code that invokes the generative AI system to capture the response. Obviously, this is only possible for systems offering an API. When code generation is feasible, it is also possible to keep a log of the outputs of generative AI systems, recording the prompts, parameters and outputs. This can be used to keep track of the quality of a given prompt over time.

Prompt optimization: As discussed in the related work (Section II), tools like PromptPerfect or Promptist aim to apply platform-dependent optimizations to improve the quality of prompts. Rather than working with the optimized prompts, it would be possible to integrate these optimizations in the prompt generation process.

Platform comparison: Different generative AI systems may produce outputs of different quality for the same prompt. It may be interesting to generate and process prompts in different platforms, either to offer all outputs as alternatives to the user or to compare their performance by automatic means.

Prompt quality and security analysis: In addition to generating code, it would be possible to generate validation artifacts, such as prompt-driven tests, to automatically assess whether the output of a generative AI tool for a given prompt fulfills the input specification. Resuming our prompt for a children's book, after generating an image we could generate another prompt for a multi-modal AI service asking whether the generated image is indeed suitable for children.

For prompts with parameters, an area of particular interest is checking whether the prompt is susceptible to *prompt injection*, *i.e.*, crafting malicious inputs to achieve goals contrary to the interests of the prompt creators, such as revealing the content of the prompt in the output.

Multi-lingual prompt generation: Given that the prompt generation process is under our control, we can generate the platform-specific prompts in languages other than English, provided that the user either (a) specifies literals in that language or (b) uses only predefined traits. While this feature has right now limited practical application as most generative AI systems are trained on datasets where English is the predominant language (and thus the performance is better when they receive prompts written in English), we see a trend towards releasing localized generative AI systems trained on other languages.

VI. Conclusions and Future Work

Prompt engineering is an emerging discipline that studies prompting strategies and their impact on the efficiency of generative AI systems. In this paper, we have discussed the benefits of applying MDE techniques to this field.

In particular, we have introduced a DSL called Impromptu for the definition of platform-independent prompts. This DSL enables features such as modular prompts, prompt customization for specific platforms and prompt documentation. Moreover, it is expressive enough to describe different types of tasks, like text-to-text or text-to-image.

As future work, we plan to expand the MDE capabilities of our framework to start supporting some of the key applications and contributions discussed in the previous section.

REFERENCES

[1] A. Mullen, N. Greene, B. Stewart, M. Halpern, and S. Barot, "Top strategic technology trends for 2022: Generative AI," 2021, Gartner report.

[2] J. Wiles. (2023) Beyond ChatGPT: The future of generative AI for enterprises. [Online]. Available: https://www.gartner.com/en/articles/beyond-chatgpt-the-future-of-generative-ai-for-enterprises

[3] M. O'Grady and M. Gualtieri, "Global AI software forecast," 2022, Forrester report.

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[5] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *arXiv preprint arXiv:2107.13586*, 2021.

[6] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[7] J. Ye, X. Chen, N. Xu, C. Zu, Z. Shao, S. Liu, Y. Cui, Z. Zhou, C. Gong, Y. Shen, J. Zhou, S. Chen, T. Gui, Q. Zhang, and X. Huang, "A comprehensive capability analysis of GPT-3 and GPT-3.5 series models," *arXiv preprint arXiv:2303.10420*, 2023.

[8] L. Reynolds and K. McDonell, "Prompt programming for large language models: Beyond the few-shot paradigm," in *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, ser. CHI EA '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3411763.3451760

[9] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," *arXiv preprint arXiv:2201.11903*, 2022.

[10] I. Cognition, "Prompt Gym," 2023. [Online]. Available: https://github.com/inspired-cognition/critique-apps/tree/main/prompt-gym

[11] S. Witteveen and M. Andrews, "Investigating prompt engineering in diffusion models," *arXiv preprint arXiv:2211.15462*, 2022.

[12] T. Z. Zhao, E. Wallace, S. Feng, D. Klein, and S. Singh, "Calibrate before use: Improving few-shot performance of language models," *arXiv preprint arXiv:2102.09690*, 2021.

[13] V. Petsiuk, A. E. Siemenn, S. Surbehera, Z. Chin, K. Tyser, G. Hunter, A. Raghavan, Y. Hicke, B. A. Plummer, O. Kerret, T. Buonassisi, K. Saenko, A. Solar-Lezama, and I. Drori, "Human evaluation of text-to-image models on a multi-task benchmark," *arXiv preprint arXiv:2211.12112*, 2022.

[14] A. Borji, "Generated faces in the wild: Quantitative comparison of Stable Diffusion, Midjourney and DALL-E 2," *arXiv preprint arXiv:2210.00586*, 2022.

[15] 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.

[16] Z. Cheng, J. Kasai, and T. Yu, "Batch prompting: Efficient inference with large language model APIs," *arXiv preprint arXiv:2301.08721*, 2023.

[17] T. Wu, M. Terry, and C. J. Cai, "AI chains: Transparent and controllable human-AI interaction by chaining large language model prompts," in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, ser. CHI '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3491102.3517582

[18] H. Strobelt, A. Webson, V. Sanh, B. Hoover, J. Beyer, H. Pfister, and A. M. Rush, "Interactive and visual prompt engineering for ad-hoc task adaptation with large language models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 1, pp. 1146–1156, 2022.

[19] S. H. Bach, V. Sanh, Z.-X. Yong, A. Webson, C. Raffel, N. V. Nayak, A. Sharma, T. Kim, M. S. Bari, T. Fevry, Z. Alyafeai, M. Dey, A. Santilli, Z. Sun, S. Ben-David, C. Xu, G. Chhablani, H. Wang, J. A. Fries, M. S. Al-shaibani, S. Sharma, U. Thakker, K. Almubarak, X. Tang, X. Tang, M. T.-J. Jiang, and A. M. Rush, "PromptSource: An integrated development environment and repository for natural language prompts," *arXiv preprint arXiv:2202.01279*, 2022.

[20] E. Jiang, K. Olson, E. Toh, A. Molina, A. Donsbach, M. Terry, and C. J. Cai, "PromptMaker: Prompt-based prototyping with large language models," in *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, ser. CHI EA '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3491101.3503564

[21] V. Liu, J. Vermeulen, G. Fitzmaurice, and J. Matejka, "3DALL-E: Integrating text-to-image AI in 3D design workflows," *arXiv preprint arXiv:2210.11603*, 2022.

[22] V. Liu and L. B. Chilton, "Design guidelines for prompt engineering text-to-image generative models," in *CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2022, pp. 1–23.

[23] K. Zhou, J. Yang, C. C. Loy, and Z. Liu, "Learning to prompt for vision-language models," *International Journal of Computer Vision*, vol. 130, no. 9, pp. 2337–2348, 2022.

[24] J. Oppenlaender, "A taxonomy of prompt modifiers for text-to-image generation," *arXiv preprint arXiv:2204.13988*, 2022.

[25] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A prompt pattern catalog to enhance prompt engineering with ChatGPT," *arXiv preprint arXiv:2302.11382*, 2023.

[26] T. Wu, E. Jiang, A. Donsbach, J. Gray, A. Molina, M. Terry, and C. J. Cai, "PromptChainer: Chaining large language model prompts through visual programming," in *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, ser. CHI EA '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3491101.3519729

[27] J. Liu, "LlamaIndex," 11 2022. [Online]. Available: https://github.com/jerryjliu/gpt_index

[28] L. Beurer-Kellner, M. Fischer, and M. Vechev, "Prompting is programming: A query language for large language models," *arXiv preprint arXiv:2212.06094*, 2022.

[29] T. Shin, Y. Razeghi, R. L. Logan, E. Wallace, and S. Singh, "AutoPrompt: Eliciting knowledge from language models with automatically generated prompts," *arXiv preprint arXiv:2010.15980*, 2020.

[30] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, W. Huang, Y. Chebotar, P. Sermanet, D. Duckworth, S. Levine, V. Vanhoucke, K. Hausman, M. Toussaint, K. Greff, A. Zeng, I. Mordatch, and P. Florence, "PaLM-E: An embodied multimodal language model," *arXiv preprint arXiv:2303.03378*, 2023.

[31] S. Huang, L. Dong, W. Wang, Y. Hao, S. Singhal, S. Ma, T. Lv, L. Cui, O. K. Mohammed, B. Patra, Q. Liu, K. Aggarwal, Z. Chi, J. Bjorck, V. Chaudhary, S. Som, X. Song, and F. Wei, "Language is not all you need: Aligning perception with language models," *arXiv preprint arXiv:2302.14045*, 2023.

[32] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, R. Ring, E. Rutherford, S. Cabi, T. Han, Z. Gong, S. Samangooei, M. Monteiro, J. Menick, S. Borgeaud, A. Brock, A. Nematzadeh, S. Sharifzadeh, M. Binkowski, R. Barreira, O. Vinyals, A. Zisserman, and K. Simonyan, "Flamingo: a visual language model for few-shot learning," *arXiv preprint arXiv:2204.14198*, 2022.

[33] "Langium," 2023. [Online]. Available: https://langium.org/