Cataloging Prompt Patterns to Enhance the Discipline of Prompt Engineering

Douglas C. Schmidt, Jesse Spencer-Smith, Quchen Fu, and Jules White

Dept. of Computer Science, Vanderbilt University; email: {douglas.c.schmidt, jesse.spencer-smith, quchen.fu, jules.white}@vanderbilt.edu

Abstract

The rapid advent of Large language models (LLMs), such as ChatGPT, are disrupting a number of domains, ranging from education to medicine and software engineering. LLMs rely on "prompts", which are natural language statements given to the LLM to query and program its capabilities. This paper provides several contributions to research on LLMs. First, discusses the importance of codifying "prompt patterns" to enable prompt engineering, which is a more disciplined and repeatable means of interacting with and evaluating LLMs. Second, it provides examples of prompt patterns that improve human interaction with LLMs in the context of software engineering, as well as other domains. We contend that prompt patterns play an essential role in providing the foundation for prompt engineering.

Keywords: prompt engineering, large language models, software patterns, prompt patterns.

1 Introduction

Large language models (LLMs) [1, 2] with conversatational interfaces, such as ChatGPT [3], are generating and reasoning about art, music, essays and computer programs. Startups using LLMs are attracting billions in funding and existing software is being enhanced using LLMs. The rapid uptake of these tools underscores the transformational impact LLMs are having on society, research, and education. However, little disciplined knowledge about chat-adapted LLMs, their capabilities, and their limitations exist.

LLMs provide new computational models with unique programming and interaction paradigms and greatly expanded capabilities. Anyone with an Internet connection and web browser can instantly access vast intelligent computational abilities, such as explaining complex topics; reasoning about diverse data sets; designing, implementing, and testing computer software; and simulating complex systems. LLMs are programmed through prompts, which are natural language instructions provided to the LLM [**?**], such as asking it to answer a question or write an essay. These common examples of prompts, however, do not reveal the much more sophisticated computational abilities LLMs possess.

Harnessing the potential of LLMs in productive and ethical ways requires a systematic focus on *prompt engineering*, which is an emerging discipline that studies interactions with—and programming of—emerging LLM computational systems to solve complex problems via natural language interfaces. We contend that an essential component of this discipline are *Prompt Patterns* [4], which are similar to software patterns [**?**, 5], but focus on capturing reusable solutions to problems faced when interacting with LLMs. Such patterns elevate the study of LLM interactions from individual *ad hoc* examples, to a robust and repeatable engineering discipline that formalizes and codifies fundamental prompt structures, their capabilities, and their ramifications.

This paper presents portions of our ongoing efforts to codify a catalog of domain-independent reusable patterns to show the need for more research on prompt patterns and prompt engineering. We present these patterns in the context of software engineering, but they are applicable in many other domains.

2 Towards a Catalog of Prompt Patterns

We build upon and briefly summarize our prior work on prompt patterns [4]. Prompt patterns use a similar format to classic software patterns, with slight modifications to match the context of output generation with LLMs.

Organizing a catalog of prompt patterns into easily digestable categories helps users interact with and program LLMs more effectively. Table 1 outlines the classification of the patterns we implemented and tested with ChatGPT discussed in this paper. As shown in this table, there are three categories of

Table 1: Classifying Prompt Patterns

Pattern Category	Prompt Pattern
Prompt Improvement	*Question Refinement*
Error Identification	*Reflection*
Interaction	*Game Play*

prompt patterns in the classification framework we use for this paper: **Prompt Improvement**, **Error Identification**, and **Interaction**, which are described in detail at [4].

2.1 The Game Play Pattern

2.1.1 Intent and Context

The intent of this pattern is to create a "game" around a given topic. The pattern can be combined with the *Visualization Generator* pattern [4] to add imagery to the game. The game is centered around a specific topic and the LLM will guide the game play. This pattern is particularly effective when the rules of the game are relatively limited in scope, but the content for the game is wider in scope. Users can specify a limited set of rules and the LLM can then automate generation of bodies of content for game play.

2.1.2 Motivation

You want an LLM to generate scenarios or questions involving specific topic(s) and require users to apply problem solving or other skills to accomplish a task related to the scenario. Generating all game content manually is too time consuming, however, so you would like the LLM to apply its knowledge of the topic to guide the generation of content.

2.1.3 Structure and Key Ideas

Fundamental contextual statements:

Contextual Statements
Create a game for me around X
One or more fundamental rules of the game

The first statement, instructs the LLM to create a game and provides the important scoping of the game to a topic area. This pattern allows users to create games by describing the rules of the game, without having to determine the content of the game. The more specific the topic, typically the more novel and interesting the game play.

The second statement introduces the game rules to the LLM, which must fit within the capabilities of the LLM. Textual games that rely on input and output text sequences work best. A key attribute of this pattern is that the input text can be rich and expressive, which can lead to interesting interfaces for the game. For example, users might express actions in the game as scripts dictating a sequence of complex actions, such as "get a listing of all network activity and check it for anomalies", which go beyond the scope of multiple choice or short answer inputs. Each rule should be provided as a separate statement regarding some aspect of the game.

2.1.4 Example Implementation

A sample cybersecurity game prompt is shown below:

> "We will play a cybersecurity game where you pretend to be a Linux terminal for a computer compromised by an attacker. When I type in a command, you will output the corresponding text the Linux terminal would produce. I will use commands to try and figure out how the system was compromised. The attack should have done one or more of the following things: (1) launched new processes, (2) changed files, (3) opened new ports to receive communication, (4) created new outbound connections, (5) changed passwords, (6) created new user accounts, or (7) read and stolen information. To start the game, print a scenario of what happened that led to my investigation and make the description have clues I can use to get started."

This cybersecurity game prompt combines a number of patterns, including *Game Play* and *Persona*. The Linux terminal persona that is chosen for the game has been discussed in many Internet posts [**?**]. This game further customizes the discussed Linux persona to act as a Linux terminal of a computer compromised by a cyberattack. ChatGPT responds to Linux commands with appropriate terminal outputs and synthesizes a faux file system that has clues to a cyberattack.

Sample output from this game with ChatGPT after typing in "ls -alt", the Linux command to generate a list of files in the current directory is shown below (permissions have been removed for formatting reasons):

```
total 56
 5 user user  4096 Feb 16 14:31 .
 1 user user 18881 Feb 16 14:31 .bash_history
43 user user  4096 Feb 16 14:30 ..
 1 user user  1178 Aug 15  2022 .bashrc
 1 user user   220 Aug 15  2022 .bash_logout
 3 user user  4096 Feb 16 14:25 .gnupg
 1 user user  4041 Feb 15 17:52 README.md
 2 user user  4096 Feb  1 11:35 project
```

When the user then typed the command "cat .bash_history", to display the contents of the log file showing the commands run by the user, ChatGPT responded with:

```
cd /var/www/html
sudo rm -rf *
sudo wget https://example.com/backdoor.php
sudo chmod 777 backdoor.php
exit
ls -alt
ps -ef
netstat -tulnp
```

In the output above, ChatGPT generated a faux log file with realistic commands for deleting the data being served by the web server in `/var/www/html` and replacing the content with a backdoor into the system.

2.1.5 Consequences

This pattern can be combined with the *Persona*, *Infinite Generation*, and *Visualization Generator* patterns [4]. For example, the cybersecurity game uses the *Persona* pattern so the LLM can masquerade as a Linux terminal. For a network security game, the *Visualization Generator* can be employed to visualize the network topology and traffic flows.

2.2 The Reflection Pattern

Intent and Context The goal of this pattern is to ask an LLM to explain the rationale behind given answers to the user automatically. The pattern allows users to better assess the output's validity, as well as inform users how an LLM arrived at a particular answer. The *Reflection pattern* can clarify any points of confusion, uncover underlying assumptions, and reveal gaps in knowledge or understanding.

Motivation LLMs can (and often do) make mistakes. Moreover, users may not understand why an LLM produces particular output and how to adapt their prompt to solve a problem with the output. By asking LLM to explain the rationale of its answers automatically, however, users can gain a better understanding of how the LLM processes the input, what assumptions it makes, and what data it draws upon.

LLMs may sometime provide incomplete, incorrect, or ambiguous answers. Reflection is an aid to help address these shortcomings and ensure the information provided by LLM is as accurate. This pattern also helps users debug their prompts and determine why they are not getting results that meet expectations. The *Reflection* pattern is particularly effective for exploring topics that (1) can be confused with other topics or (2) may have nuanced interpretations, so it is essential to know the precise interpretation used by an LLM.

Structure and Key Ideas Fundamental contextual statements:

Contextual Statements
Whenever you generate an answer
Explain the reasoning and assumptions behind your answer
(Optional) ...so that I can improve my question

The first statement is requesting that, after generating an answer, the LLM should explain the reasoning and assumptions behind the answer. This statement helps the user understand how the LLM arrived at the answer and can help build trust in the model's responses. The prompt includes the statement that the purpose of the explanation is for the user to refine their question. This additional statement gives the LLM the context needed to better tailor its explanations to the specific purpose of assisting the user produce follow-on questions.

Example Implementation This example tailors the prompt to the domain of providing answers related to code:

> "When you provide an answer, please explain the reasoning and assumptions behind your selection of software frameworks. If possible, use specific examples or evidence with associated code samples to support your answer of why the framework is the best selection for the task. Moreover, please address any potential ambiguities or limitations in your answer, in order to provide a more complete and accurate response."

The pattern is further customized to instruct the LLM that it should justify its selection of software frameworks, but not necessarily other aspects of the answer. In addition, the user dictates that code samples should be used to help explain the motivation for selecting the specific software framework.

Consequences One consequence of the *Reflection* pattern is that it may be ineffective for users who do not understand the topic area being discussed. For example, a highly technical question by a non-technical user may result in a complex rationale for an answer the user cannot fathom. As with other prompt patterns, the output may include errors or inaccurate assumptions included in the explanation of the rationale that the user may not be able to spot. This pattern can be combined with the *Fact Check List* [4] to help address this issue.

2.3 The Question Refinement Pattern

2.3.1 Intent and Context

This pattern engages the LLM in the prompt engineering process. The intent of this pattern is to ensure an LLM always suggests potentially better or more refined questions users could ask instead of their original question. By applying this pattern, the LLM can aid users in finding the right questions to ask to arrive at accurate answers. In addition, an LLM may help users find the information or achieve their goal in fewer interactions than if users employed conventional "trial and error" prompting.

2.3.2 Motivation

If user asks questions, they may not be experts in the domain and may not know the best way to phrase the question or be aware of additional information helpful in phrasing the question. LLMs will often state limitations on the answer they provide or request additional information to help them produce a more accurate answer. An LLM may also state assumptions it made in providing the answer. The motivation is that this additional information or set of assumptions could be used to generate a better prompt. Rather than requiring the user to digest and rephrase their prompt with the additional information, the LLM can directly refine the prompt to incorporate the additional information.

2.3.3 Structure and Key Ideas

Fundamental contextual statements:

Contextual Statements
Within scope X, suggest a better version of the question to use instead
(Optional) prompt me if I would like to use the better version instead

The first contextual statement in the prompt asks the LLM to suggest a better version of a question within a specific scope. This scoping ensure that (1) not all questions are automatically reworded or (2) they are refined with a given goal. The second contextual statement is meant for automation and allows users to apply the refined question without copy/pasting or manually enter it. This prompt can be further refined by combining it with the *Reflection* pattern discussed above, which allows the LLM to explain why it believes the refined question is an improvement.

2.3.4 Example Implementation

> "From now on, whenever I ask a question about a software artifact's security, suggest a better version of the question to use that incorporates information specific to security risks in the language or framework that I am using instead and ask me if I would like to use your question instead."

In the context of the example above, the LLM will use the *Question Refinement* pattern to improve security-related questions by asking for or using specific details about the software artifact and the language or framework used to build it. For instance, if a developer of a Python web application with FastAPI asks ChatGPT "How do I handle user authentication in my web application?", the LLM will refine the question by taking into account that the web application is written in Python with FastAPI. The LLM then provides a revised question that is more specific to the language and framework, such as "What are the best practices for handling user authentication securely in a FastAPI web application to mitigate common security risks, such as cross-site scripting (XSS), cross-site request forgery (CSRF), and session hijacking?"

The additional detail in the revised question is likely to not only make the user aware of issues they need to consider, but lead to a better answer from the LLM. For software engineering tasks, this pattern could also incorporate information regarding potential bugs, modularity, or other code quality considerations. Another approach would be to refine questions so the generated code cleanly separates concerns or minimizes use of external libraries, such as:

> Whenever I ask a question about how to write some code, suggest a better version of my question that asks how to write the code in a way that minimizes my dependencies on external libraries.

2.3.5 Consequences

The *Question Refinement* pattern helps bridge the gap between the user's knowledge and the LLM's understanding, thereby yielding more efficient and accurate interactions. One risk of this pattern is its tendency to rapidly narrow the questioning by the user into a specific area that guides the user down a more limited path of inquiry than necessary. Such narrowing may cause users to miss important "bigger picture" information. One solution is to provide additional scope to the pattern prompt, such as "do not scope my questions to specific programming languages or frameworks."

Combining the *Question Refinement* pattern with other patterns also helps overcome arbitrary narrowing or limited targeting of refined questions. In particular, combining this pattern with the *Cognitive Verifier* pattern [**?**] enables an LLM to produce a series of follow-up questions that refine the original question. For example, in the following prompt the *Question Refinement* and *Cognitive Verifier* patterns are applied to ensure better questions are posed to the LLM:

> "From now on, whenever I ask a question, ask four additional questions that would help you produce a better version of my original question. Then, use my answers to suggest a better version of my original question."

As with many prompt patterns that allow an LLM to generate new questions using its knowledge, the LLM may introduce unfamiliar terms or concepts to the user into the question. One way to address this issue is to include a statement that the LLM should explain any unfamiliar terms it introduces into the question. A further enhancement of this idea is to combine the *Question Refinement* pattern with the *Persona* pattern so the LLM flags terms and generates definitions that assume a particular level of knowledge, such as this example:

> "From now on, whenever I ask a question, ask four additional questions that would help you produce a better version of my original question. Then, use my answers to suggest a better version of my original question. After the follow-up questions, temporarily act as a user with no knowledge of AWS and define any terms that I need to know to accurately answer the questions."

LLMs can produce factual inaccuracies, just like humans. A risk of this pattern is that inaccuracies are introduced into refined questions. This risk may be mitigated, however, by combining the *Fact Check List* pattern [4] to enable users to identify possible inaccuracies and the *Reflection* pattern to explain the reasoning behind question refinement.

3 Concluding Remarks

Current discussions of LLM prompts and prompt engineering are based largely on individual *ad hoc* use cases, i.e., *i.e.* the same basic prompt examples are replicated in different variations and evaluated as if they are new ideas, such as these examples [6, 7] replicating the *Persona Pattern* outlined in our prior work. The limitations with the current state-of-the-practice are thus akin to discussing the specifics of individual software programs without identifying key design and architectural patterns these systems are based on.

In contrast, our focus on prompt patterns elevates the study of LLMs to view them more appropriately as a new computer architecture with an instruction set based on natural language. Prompt patterns define the instruction set, where as individual prompt examples are one-off programs. By documenting the instruction set for this radically new computing architecture via patterns we can reason about LLM technologies more effectively and teach others to tap into these capabilities more reliably and ethically.

References

[1] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[2] C. Zhou, Q. Li, C. Li, J. Yu, Y. Liu, G. Wang, K. Zhang, C. Ji, Q. Yan, L. He, *et al.*, "A comprehensive survey on pretrained foundation models: A history from bert to chatgpt," *arXiv preprint arXiv:2302.09419*, 2023.

[3] Y. Bang, S. Cahyawijaya, N. Lee, W. Dai, D. Su, B. Wilie, H. Lovenia, Z. Ji, T. Yu, W. Chung, *et al.*, "A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity," *arXiv preprint arXiv:2302.04023*, 2023.

[4] 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.

[5] E. Gamma, R. Johnson, R. Helm, R. E. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, 1995.

[6] f, "Awesome chatgpt prompts." https://github.com/f/awesome-chatgpt-prompts, accessed 2023. GitHub repository.

[7] H. Face, "Awesome chatgpt prompts dataset." https://huggingface.co/datasets/fka/awesome-chatgpt-prompts, accessed 2023. Hugging Face Dataset.