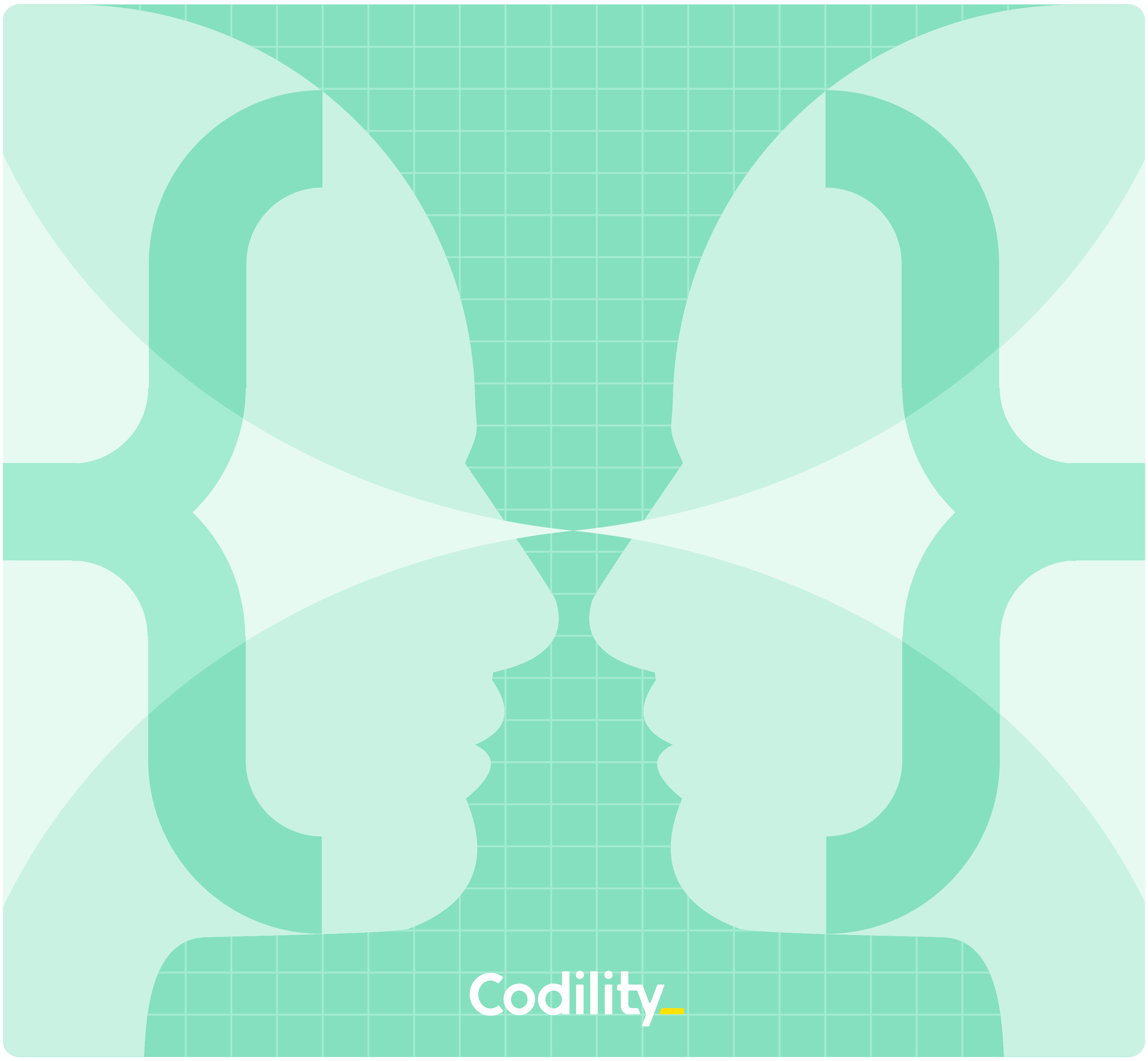
WHITE PAPER

Harnessing the Power of Generative AI: A Guide to Best Practices in Software Engineering

By James Meaden, Assessment Scientist, Taylor Sullivan, PhD and Neil Morelli, PhD



Harnessing the Power of Generative AI: A Guide to Best Practices in Software Engineering

James Meaden
Assessment Scientist

Taylor Sullivan
PhD

Neil Morelli
PhD

TABLE OF CONTENTS

1. Understanding Generative AI 01
1.1. A Revolution in Software Engineering 01
2. Human-AI Collaboration 02
2.1. Prompt Engineering 03
2.2. Writing Effective Prompts 04
2.3. Multiple Techniques for Prompt Writing 05
2.4. Multiple Methods for Prompt Writing 07
2.5. Prompt Engineering Best Practices 08
3. Choosing a Generative AI Model 09
3.1. Commercial Versus Open-Source Models 10
3.2. Factors to Consider When Reviewing Specific Models 10
Commercial Models 11
Open-Source Models 11
3.3. Consider Fine-Tuning a Generative AI Model 11
4. Developing AI Systems 12
4.1. Frameworks for Developing AI Systems 13
4.2. Optimizing System Intelligence with AI Chain Engineering 13
4.3. Prioritizing User Interaction in AI Systems 14
4.4. Autonomous Learning Systems with Human Oversight 14
Summary 15
References 16

1. Understanding Generative AI

Artificial Intelligence (AI) is a broad category of technology systems that combine machine learning and data processing to perform tasks that have typically required human intelligence, such as language translation, speech recognition, and vision-based object identification. Generative AI refers to a subset of these technology systems that utilize deep learning algorithms to identify and learn from important patterns in large datasets.

However, generative AI is not just another tool; when used correctly it has the potential to significantly boost the productivity of software engineering teams.

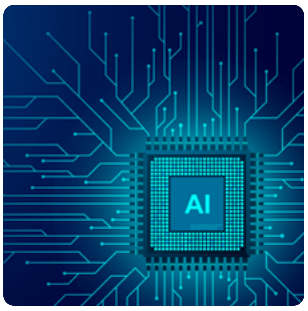
What distinguishes generative AI from other forms of AI is its ability to apply these learned patterns to create original and professional-level content, such as short stories, photorealistic images, songs, high-fidelity videos, and even code for computer programs. Generative AI's ability to create code that satisfies a user's requirements makes it a powerful new tool in the software engineer's tool bag.

However, generative AI is not just another tool; when used correctly it has the potential to significantly boost the productivity of software engineering teams.

It is becoming clear that effectively harnessing generative AI will become a strategic imperative for software engineering teams. Unfortunately, there is a lack of guidance for engineering team leaders regarding how this technology can be used to increase their teams' performance, productivity, and work engagement. This paper addresses this gap by providing information on three important aspects of generative AI in relation to software engineering: 1) how to collaborate with AI, 2) which AI to collaborate with, and 3) how to build effective AI systems.

We hope the information in this paper provides clarity and stimulates some ideas for using generative AI to optimize the outputs of your engineering teams.

1.1. A Revolution in Software Engineering



Generative AI has the potential to transform many industries as organizations stand to benefit from its ability to automate information retrieval and content generation. For example, in the education industry, librarians [1] and academic researchers [2] can use generative AI for tasks ranging from automated information cataloging to summarizing volumes of peer-reviewed papers. In the medical sector, generative AI can assist in predicting protein structures, which can accelerate drug discovery [3]. In the Architecture, (Civil) Engineering, and Construction (AEC) industry, generative AI can automate compliance checking by comparing building design specifications against safety codes [4]. Unsurprisingly, generative AI is also poised to disrupt the content creation industry by enabling the development of increasingly sophisticated artwork such as poetry, music, and multiple forms of visual art, and it will be able to do this at scale. These are just a few examples of how generative AI could transform entire industries over the coming years.

In the technology industry specifically, generative AI is already having a strong impact and has enormous potential to revolutionize long-established practices and methods.

For example, AWS CodeWhisperer, powered by generative AI, provides engineers with code recommendations and has been found to significantly multiply a software engineer's productivity, often by accelerating the initial stages of code development [5]. Using tools like this allows software engineers to spend less time on routine code development tasks and more time applying their expertise to solve more complex, challenging, and engaging problems. This freeing up of cognitive capacity and engineering expertise is likely to lead to more innovative and optimized software solutions.

While generative AI tools come with great potential to increase the efficiency and productivity of engineering teams, the ability to reap this potential is limited to those who can collaborate with them effectively. For example, while generative AI models can successfully write code for fairly simple tasks, they tend to struggle with more complex coding tasks [6][7]. While this may change as generative AI models become increasingly more sophisticated, we strongly believe that human engineering expertise will remain a crucial element when using generative AI to assist in developing complex software programs .

Because of this need for human-AI collaboration, engineering teams that can optimally combine human expertise with AI assistance will achieve superior levels of performance and productivity. To help engineering teams do this, we conducted a comprehensive and multidisciplinary review of peer-reviewed research on the techniques, methods, and best practices for collaborating with generative AI models. This paper summarizes our findings.

2. Human-AI Collaboration

Broadly speaking, there are two ways that an engineer will collaborate with generative AI models to produce code (or other outputs, such as code comments or information about a specific package).

The first is via a method similar to the autocomplete functionality in predictive texting. In this method, the engineer does not necessarily need to provide any explicit instructions to the generative AI model for it to provide code suggestions. Instead, the generative AI model takes existing code that precedes or follows the line of code on which the engineer is currently working and uses this information to inform its code suggestions. An example of this kind of auto-complete generative AI tool is Github Copilot.

The second way an engineer will collaborate with generative AI to produce code is to provide direct instructions, known as **prompts**, to guide the generative AI model to provide an output that satisfies the user's request. Interacting with generative AI in this way often requires an iterative approach.

It starts with an initial prompt given by an engineer, which is followed by the AI's response. Based on this response, the engineer may refine their prompt to guide the AI toward a more desirable output. For example, an engineer might start with a prompt like, "Create a Python function that loops through multiple lists of integers and outputs the mean of each list." Then, the engineer may provide feedback to the AI on how the function could be improved by specifying certain conditions that must be met. This process continues until the engineer is satisfied with the code provided (sometimes, slight manual modifications may be required).

This type of interaction with generative AI is known as **prompt engineering**, which is "an emerging field that requires creativity and attention to detail. It involves selecting the right words, phrases, symbols, and formats that guide the model in generating high-quality and relevant texts" [8]. An example of this kind of interaction-based (or chat-based) generative AI tool is OpenAI's Advanced Data Analysis model (formerly referred to as Code Interpreter).

This type of interaction with generative AI is known as prompt engineering, which is "an emerging field that requires creativity and attention to detail. It involves selecting the right words, phrases, symbols, and formats that guide the model in generating high-quality and relevant texts" [8].

Understanding prompt engineering is important regardless of whether an engineer uses an auto-complete or an interaction-based generative AI tool. Prompt engineering is naturally a key process when using interaction-based tools; however, it is also built into auto-complete tools as the previously written code or comments essentially act as the prompt.

Given the importance of prompt engineering in collaborating with generative AI, we now turn to an overview of the various types, methods, and best practices for prompt engineering.

2.1. Prompt Engineering

Effective prompt engineering starts with understanding the capabilities and limitations of the specific generative AI model with which an engineer is working. With an ever-increasing number of generative AI models available, each one trained under different conditions of model architecture, parameters, and training data, engineers will need to be aware that each model is likely to respond to different prompts and different styles of prompts in different ways - what works well for one generative AI model will not necessarily work well for another model. Therefore, when collaborating with a new generative AI model, it is essential for engineers to quickly learn the methods and techniques that will lead to optimal outputs for that particular model.

Information that is important to know for each model includes the format of inputs (e.g., Can you upload documents? Can special characters be used to store variables, for example "[variable 1]"?) and outputs (e.g., Which languages will it provide code for? Can it give an output in markdown format?) and the prompt types and methods to which it responds best (discussed in the following sections).

For starters, when an engineer is working with a new generative AI model designed to write code, they should begin by building their understanding of the coding languages the generative AI has been trained on (and ideally, the percent of the training data that represents their chosen language), the types of and complexity of code it can successfully create, and the potential issues such as how often the generative AI model is likely to provide ineffective code or code with potential security issues [9]. Learning this information and more can be achieved by reviewing publicly available documents and data, discussing with colleagues, or via trial-and-error. A combination of these learning methods typically leads to the best results.

Once an engineer is up-to-speed and effectively collaborating with the new generative AI model, they will need to stay informed regarding any changes to the model or version updates that may impact their method of collaboration. The pace of iteration and updates to generative AI models can be rapid, and engineers should monitor for version updates that, while bringing improved model sophistication and capabilities, may reduce model performance when using the engineer's current methods. It is highly recommended that engineering teams keep track of changes to generative AI models and the associated implications for their working methods.

While each generative AI model is unique, and engineers will need to understand these differences and adapt their interactions accordingly, there are some approaches and best practices that tend to generalize across models. These approaches and best practices, summarized below, cover not only what to include in a prompt but also how to generate the prompt and how to present the prompt to the generative AI model.

2.2. Writing Effective Prompts

Prompt writing involves crafting prompts or commands to guide the AI's content generation process. These prompts serve as the input the AI uses to produce its output, making them a critical aspect of how users interact with generative AI. The significance of prompt writing in the context of generative AI cannot be overstated, as how a prompt is formulated can significantly influence the AI's output. For instance, providing more context in the prompt or asking the AI to think step by step can often lead to more detailed and accurate outputs. Conversely, vague or ambiguous prompts can lead to inadequate or incorrect outputs. **Thus, the ability to craft effective prompts is a crucial skill to learn for engineers collaborating with generative AI**.

Prompt writing is an emerging skill set, and much remains to be understood regarding what makes one prompt more effective than another. For this reason, prompt writing has been described as a trial-and-error process [10], and it is not always clear which aspects of a prompt will most influence a generative AI's output. One critical insight that has been established is that humans and generative AI models process information differently and, therefore, a prompt that appears clear and concise to an engineer may not necessarily result in the intended output from a generative AI model.

Humans and generative AI models process information differently and, therefore, a prompt that appears clear and concise to an engineer may not necessarily result in the intended output from a generative AI model.

Another consistent finding is that effective prompt writing requires clear instructions and specifications. Consider a scenario where a user asks a generative AI model to create a short story. An ineffective prompt would be, "Write a short story about a software engineer starting their first job" – this prompt is too vague and doesn't provide enough context. On the other hand, a more effective prompt would be, "Write an inspiring short story of around 500 words about a person starting their first day at their first job as a software engineer after finishing college, focusing on the person's feelings of excitement and possibilities for their future." This prompt provides a more specific objective with clear context and constraints. One caveat to this general rule is that shorter and less specific prompts can work well in certain situations, for example when a model has been trained or fine-tuned for a particular use case (see Section 3.3).

2.3. Multiple *Techniques* for Prompt Writing

Prompt writing is not a one-size-fits-all process; instead, it includes a variety of techniques. These can be broadly categorized into six types:

1 Zero-shot prompting

2 Few-shot prompting

3 Chain of Thought (CoT) prompting

4 Prompt ensembling

5 Metaprompting

6 Tree of Thought (ToT) prompting

Zero-shot and few-shot prompting are the most basic. With **zero-shot prompting**, the user's prompt contains task instructions for a generative AI model without specific examples regarding the output's expected format. On the other hand, **few-shot prompting** includes a small set of examples that guide and constrain the generative AI's outputs. Some studies have found that few-shot prompting enhances performance on tasks such as code summarization, especially when developers use samples from the same or similar projects [11]. However, other studies have found that few-shot prompting does not always enhance performance over zero-shot prompting [12].

Chain of Thought (CoT) and prompt ensembling are more advanced techniques. CoT prompting involves instructing the generative AI model to break down a problem into smaller sub-tasks, then provide a rationale behind decisions made for each sub-task before ultimately providing a final output. This prompting method has been found to increase performance across various tasks, including logical reasoning tasks [13].

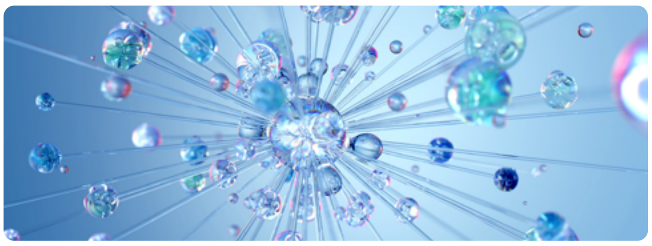
The **prompt ensembling** technique borrows from the concept of ensemble models in machine learning, in which the outputs from multiple models are combined (often aggregated) to arrive at a more accurate final output. With prompt ensembling, the final output from a generative AI model results from multiple intermediate outputs, each with its unique prompt. Prompt ensembling aims to increase the accuracy and usability of model outputs by trying similar variants of an original prompt and combining relevant aspects of the multiple outputs of these prompt variants into a final output. This final output can seek to optimize for similarity or creativity in the intermediate outputs. Depending on the method used, prompt ensembling can increase the stability and reliability or the creativity and originality of the model outputs.

Currently, the most sophisticated prompt writing techniques include metaprompting and Tree of Thought (ToT) prompting.

Metaprompting is an innovative approach where the user asks the generative AI model to create its own prompts to complete a specified task [12]. These AI-generated prompts can then be evaluated against an objective [14], paving the way for self-learning systems.

For example, an engineer wants to write a prompt that will optimize the code review accuracy for a specific generative AI model for a specific project. Using the metaprompting technique, the engineer would write a prompt that instructs the generative AI model to output a prompt that would optimize its ability to review code, perhaps specifying some context about the type of code, the project, and what the model should be optimizing for. In this scenario, let's say the engineer is optimizing for runtime. The engineer will then take the initial output from the generative AI model - the optimized code review prompt - and then use that prompt to instruct the model to conduct a code review (typically, but not necessarily, using the same model). After suggested revisions from the code review are made, the engineer stores the text from the "optimized code review prompt" and the runtime of the resulting code.

After a few iterations of this process, the engineer can then provide the generative AI model with a series of paired prompt text and runtime values and instruct the model to use this information to generate another "optimized code review prompt" that seeks to minimize the associated runtime value. This process continues until the engineer is satisfied with the resulting performance of the code review prompt. Note how this specific example of metaprompting incorporates principles from supervised machine learning.



Tree of Thought (ToT) prompting also utilizes a principle from machine learning, that of tree-based supervised learning. ToT extends upon the CoT prompting technique by instructing the generative AI model to consider multiple alternative approaches at each step (i.e., each sub-task) in the problem-solving process. At each step, the model first describes the sub-task, provides numerous options for addressing the sub-task, and then selects one of these options before moving on to the next step, and the process repeats until it arrives at a solution. Critically, the ToT technique incorporates instructions for the model to backtrack to previous steps and to reconsider the available options if the originally (or subsequently) chosen problem-solving path turns out to be suboptimal [15]. This technique enables the consideration of multiple stepwise approaches to solving a specific problem.

Zero-shot prompting:
providing a generative AI model task instructions without specific examples regarding the output's expected format.

Few-shot prompting:
providing a generative AI model task instructions along with a small set of examples that guide and constrain the generative AI's outputs.

Metaprompting:
guiding a generative AI model through the process of creating its own prompts to complete a task.

Prompt ensembling:
generating multiple outputs from a generative AI model by providing similar variants of an original prompt, and then instructing the generative AI model to consolidate information from the multiple outputs.

Chain of Thought (CoT) prompting:
guiding a generative AI's final output by instructing the model to break down a problem into smaller sub-tasks and to then make a decision on the best way to complete each subtask along with a rationale behind each decision.

Tree of Thought (ToT) prompting:
guiding a generative AI's final output by instructing the model to break down a problem into smaller subtasks, to consider multiple solutions for each subtask, to make a decision on the best way to complete each subtask, and to backtrack to previous subtask decisions when necessary.

Let us return to our previous example of the engineer who wants to create an optimized code review prompt. Applying the ToT to this objective may involve the engineer instructing the model to do the following:

1.

Break down the task of creating an optimized prompt for code review into multiple sub-tasks, which might result in the AI outputting sub-tasks such as: "1. Ensure clarity of objective statement", "2. List key considerations to achieve objective statement", "3. Create a first draft of prompts that incorporate key considerations", and "4. Review and revise first drafts".

2.

List multiple solutions for the first sub-task and select one of those solutions (while providing a rationale for that decision), and then move on to the second sub-task and repeat the process until arriving at a recommended "optimized code review prompt" as the final output for that particular decision path.

3.

Review the output of this path and the decisions made at each sub-task step and consider whether to explore alternate paths to improve the final output.

The unique capability of the ToT prompting method is that it can cover a larger area of the potential solution space by considering multiple possible decision paths. In principle, this technique could be repeated in a way that is conceptually similar to random forest models in machine learning. Doing so would involve instructing the generative AI to repeat the ToT method multiple times, with each "tree" having unique decision "branches," and the final recommendations from each tree could be used as a shortlist of solutions from which the generative AI model can choose.

As described above, there are multiple techniques to prompt writing, and they vary significantly in their complexity and capabilities. The more basic prompt writing techniques often suffice for simple software engineering tasks, such as boilerplate code creation. On the other hand, more sophisticated techniques may be required for more advanced tasks or tasks that require a solution tailored to a particular use case (e.g., optimizing prompts for a specific project).

2.4. Multiple *Methods* for Prompt Writing

In addition to different prompt writing techniques, various methods can be used to execute these techniques. These include the following [16] (in order of increasing sophistication):

1. **Manual construction:** Prompts are developed by humans via one of three methods: a) static prompts that are predefined by experts, b) template-based prompts that contain static text along with "fill in the blank" areas where users enter additional text, and c) free-form prompts that the user creates without any constraints.
2. **Language model generation:** Prompts are developed entirely or partially by a generative AI model to meet the objective specified in a user's initial prompt.

3. **Retrieval-based prompt:** Prompts include information from an external source, such as an organization's internal documents or published academic literature.
4. **Prompt learning:** Prompts are developed iteratively via a supervised machine learning model to optimally satisfy a user's predefined objective.

Notice that we included some of these prompting methods (e.g., language model generation and prompt learning) in the previous section as examples of how prompt writing techniques might be implemented. Each prompting method has distinct advantages and is suitable for different scenarios.

In summary, effectively selecting from these different prompting techniques and methods based on the nature of the task is essential for harnessing the potential of generative AI. We now turn to some additional best practices for prompt engineering that supplement these techniques and methods.

2.5. Prompt Engineering Best Practices

While effective prompt writing often involves experimentation and iteration [17], generalizable best practices for effectively interacting with generative AI models have begun to surface from the scientific community. Highlights are summarized below.

Prompt Engineering Best Practices:

1. **Structure your prompt**
2. **Include relevant context**
3. **Experiment with positioning**
4. **Document and iterate**

1. **Structure your prompt**
   Underlying all prompt writing best practices is the core principle that interacting with generative AI models should be approached systematically.
   While we use natural language (i.e., human language) to communicate with generative AI, it is crucial to remember that these models do not interpret language the same way humans do [18]. Long and overly comprehensive descriptions can limit a generative AI's ability to recall the most relevant information in a prompt when constructing its output, potentially reducing its performance [19]. In addition, current generative AI models have a limited amount of text that they can process in a prompt. When crafting prompts, users need to be mindful of this limit to ensure effective utilization of the model's capabilities. As models advance, prompt size limits typically increase, so it's best to consult official documentation or community resources for updated specifications. Generally speaking, prompts should be concise, clear, and provide an objective. Structured frameworks for writing effective prompts highlight these aspects and others, including Conciseness, Logic, Explicitness, Adaptability, and Reflectiveness (CLEAR) [20].

2. **Include relevant context**
   Research has found that carefully crafted prompts can substantially improve the code generation process [21].
   For software engineers, this means writing prompts with rich programming context. For example, an engineer might include relevant classes, member variables, and functions in a prompt. Enclosing code snippets in triple quotes (''') can help some generative AI models better comprehend code blocks within markdown syntax.

3. **Experiment with positioning**
   The position of a specific word or block of text within a prompt also significantly influences the output [22].
   For example, researchers have found that performance is often higher when the most relevant and important information occurs at the beginning or end of a prompt [23]. Prompt position optimization should be considered during prompt creation and iteration. Optimizing the position of key text phrases within a prompt can be achieved systematically via the prompt learning method described earlier. However, this method can take time to develop, and less sophisticated prompt optimization requires user experimentation and iteration through a trial-and-error process. Remember that once a prompt is optimized for a particular generative AI model, it may need to be revised as new versions of the model are released. Also, the optimized prompt may not work well with other generative AI models.
4. **Document and iterate**
   Engineering teams can benefit from developing an in-house database containing prompts and associated outputs.
   This database can serve multiple purposes: it can act as a reference for other users, provide a record for documentation purposes, and even contribute to the development of supervised machine learning models for prompt optimization [24] (including the optimization of prompt position, described in the preceding paragraph).

As the trajectory and ultimate level of sophistication of generative AI's capabilities are actively being discussed and debated, a high degree of uncertainty remains. Some propose that as generative AI models evolve, they might reach a level where they can understand and fulfill a user's intended objectives from rudimentary and relatively unstructured prompts [25]. If this level of sophisticated understanding is achieved, skill in prompt writing may become less critical. However, at the current time, prompt writing is an essential skill for effectively collaborating with generative AI tools.

Having provided an overview of how to interact with - better yet, how to collaborate with - generative AI, we next turn to the question:

How do I know which generative AI models I should collaborate with?

3. Choosing a Generative AI Model

The number of generative AI models available to assist software engineering teams is rising sharply. Organizations such as OpenAI, Google, Microsoft, and Facebook are releasing increasingly sophisticated models at an increasing pace. In addition, an active open-source community is led by organizations such as Hugging Face. This platform provides access to a growing number of open-source machine-learning models and currently has over 19,000 text-generation models available. With such an expansive landscape of generative AI models, engineering leaders face a crucial decision: Which generative AI models should the team use?

3.1. Commercial Versus Open-Source Models

The first step in answering this question is to consider whether commercial generative AI models such as Github Copilot or open-source models such as Alpaca are better suited to the needs of your team, specific projects, and the organization's IT infrastructure and security requirements.

A key benefit of commercial models is that they come equipped with advanced moderation filters, which help ensure the generated content is relevant and adherent to established benchmarks. The promise of scalability, fortified by robust APIs, makes them a good fit for large engineering teams. Commercial models also offer potential cost and resource savings as the vendor handles the model maintenance and continual improvement.

However, these benefits come with caveats. Commercial models can be cost-prohibitive, especially for smaller entities or startups. Moreover, the structured environment, while beneficial in many respects, can sometimes stifle innovation, given the limited scope for customization. This rigidity can also lead to potential vendor lock-in, restricting future adaptability.

In contrast, open-source models provide cost-effectiveness and malleability. The possibility of reduced cost combined with the freedom to adapt and modify these models makes them especially appealing for teams that value flexibility and innovation. The global community of open-source developers continually refines these models, drawing from diverse perspectives and expertise. However, they come with their own set of challenges. Scalability, especially for larger applications, might become a performance bottleneck.

Additionally, while 'open-source' might suggest unrestricted use, understanding licensing nuances is paramount, especially for commercial endeavors.

To summarize, several critical factors influence the choice between commercial and open-source models, including the organization's size, industry regulations, financial resources, and IT infrastructure. Established corporations might favor the reliability and structured environment of commercial offerings, valuing the long-term support and maintenance they provide. In contrast, with their inherent agility and often constrained budgets, startups might lean toward open-source alternatives, valuing their adaptability and potential for fostering innovation. Central to this decision matrix is the team's expertise. Teams with experience in generative AI model development and maintenance might thrive in the open-source environment. In contrast, teams with less experience in this area might seek the predictability of commercial models.

3.2. Factors to Consider When Reviewing Specific Models

Once the decision to use commercial or open-source models (or a combination of both) has been made, the next step is to review the specific models available. Presented below are some key factors to consider during this process.

Commercial Models

When reviewing commercial models, it is important to evaluate the vendor's position and trajectory compared to their competitors in the generative AI landscape. A good approach is to focus on vendors with a history of staying at the forefront of innovation while prioritizing safety and ethical standards in their models. In addition to information about the vendor's generative AI models, it's also useful to review information about their support services and general reputation among customers and the broader AI community. Another obvious and perhaps primary consideration is cost, which includes usage fees and other potential costs such as licensing flexibility, scaling costs, and any hidden fees. Integration capabilities—especially the model's ability to meld seamlessly with existing systems—are another critical factor to consider.

While commercial models typically come with more constraints, some vendors offer avenues for limited customization, a feature that can be invaluable for specific project requirements. Finally, model version control and access to archived models have become crucial considerations, as some research has found that model updates can reduce performance in certain domains [26].

Open-Source Models

The world of open-source models, replete with flexibility, brings its own set of considerations. A thriving community around a model not only indicates its reliability and future update prospects but also provides an abundance of user feedback. This feedback can be a potential source of information on the model's performance across a wide range of scenarios. Models with comprehensive documentation, tutorials, and user-generated content can ease the adoption process, which is important given the potentially steep learning curve associated with using open-source models. Official performance benchmarks, often available through community forums or third-party evaluations, can provide empirical insights into the model's capabilities.

Alongside these benefits, engineering leaders need to be vigilant about security considerations. Being aware of potential vulnerabilities and ensuring the model is safe for the intended application is essential. In addition, running and training an open-source model can be costly and require significant computing power.

3.3. Consider Fine-Tuning a Generative AI Model

Fine-tuning involves adapting a pre-trained model to cater to specific needs or domains, building upon the foundational capabilities it already possesses [27].

A final consideration is whether to fine-tune the selected model(s) to your specific use case. **Fine-tuning** involves adapting a pre-trained model to cater to specific needs or domains, building upon the foundational capabilities it already possesses [27].

For example, a software engineering team may fine-tune an existing generative AI model by training it on their entire code base, resulting in a model that can suggest code in the language, frameworks, and style of the existing code base while also using the correct variable types and names. This process is typically used when the task requires specific knowledge or when the data used in pre-training differ significantly from the data the model will be generating [28].

Fine-tuning has numerous benefits. It allows for task-specific improvements in model performance and enables the model to generate more relevant and accurate outputs. However, fine-tuning must be approached carefully and with expertise. One of the risks associated with fine-tuning is that the new model becomes too specialized on the training data, resulting in outputs that highly resemble code in the training data but may not fit the user's current needs (a concept known as **overfitting**).

When developing a fine-tuned generative AI model, you should pay consideration to selecting an appropriate base model and to the data on which this model will be trained (i.e., fine-tuned). In addition, the team's expertise, both in terms of AI capabilities and domain knowledge, is essential to navigate the intricacies of fine-tuning successfully. Engineering leaders should also be mindful of potential risks, such as **model drift**, where the model's overall performance begins to decline over time. Thus, the process doesn't end with fine-tuning. Continuous evaluation, benchmarked against clear metrics, ensures the model remains aligned with its objectives.

In summary:

The choice of which generative AI tool(s) to invest in is a strategic decision that will shape the long-term success of engineering teams and broader risks and potential opportunities for the organization.

The foundational decision between commercial models, known for their robustness and support, and open-source modes, which provide flexibility and cost-effectiveness, will depend upon specific needs, resources, and operating environments. Engineering leaders should evaluate the trade-offs of each option and consider factors such as industry regulations, systems architecture, data security policies, expected performance enhancements, resource availability, and budget, among other factors discussed above.

4. Developing AI Systems

So far, we have covered how to collaborate with generative AI and how to choose which generative AI models to collaborate with. In this final section, we briefly present some examples of more advanced and innovative approaches for leveraging generative AI to increase the performance and productivity of engineering teams by incorporating generative AI models into **intelligent systems**. These systems are designed to go beyond the raw capabilities of generative AI models to produce a harmonious interplay between AI and human experts while ensuring optimized, efficient, and safe outcomes.

4.1. Frameworks for Developing AI Systems

Engineering teams embarking on the journey of developing AI systems should prioritize three key concepts: effectiveness, efficiency, and safety. Central to this is understanding the concept of **"generative variability"** — unlike traditional algorithms that have deterministic outputs, generative AI model outputs are diverse and vary in quality, quantity, and character [29].

Unlike traditional algorithms that have deterministic outputs, generative AI model outputs are diverse and vary in quality, quantity, and character [29].

Undeniably a strength of generative AI models, this variability also demands vigilance. AI systems should be architected to accommodate and, when necessary, regulate this diversity, ensuring it aligns with the intended application. Engineers should consider this variability when designing systems that interact with or rely on generative AI outputs.

Safety considerations are of utmost importance when developing AI systems. It has been said that prevention is better than cure, and this adage is particularly relevant with respect to AI. Designing against potential harm is not just a best practice but an ethical responsibility [29]. While transformative, the multifaceted capabilities of generative AI come with tremendous risks. Engineers should strive to preempt potential misuse, integrating safeguards that deter harmful or inappropriate content generation. And in scenarios where complete prevention of misuse is not possible, robust mechanisms to swiftly detect and rectify such deviations become essential.

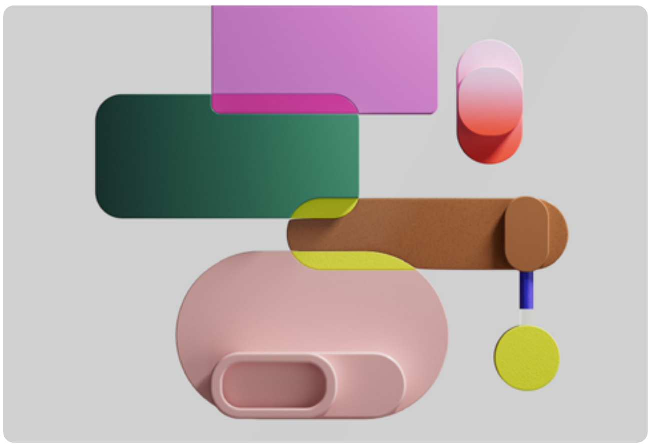
4.2. Optimizing System Intelligence with AI Chain Engineering

Harnessing the power of AI chain engineering can exponentially amplify the productivity of engineering teams. Such systems are emblematic of the potential power of fully optimized human-AI collaboration. By orchestrating the interaction between multiple generative AI models, each excelling in its own domain, a cohesive human-directed system can be developed to deliver superior outcomes.

Consider an integrated system in which generative AI models play distinct roles — a prompt refiner, a software engineer, and a product designer. As these models engage in iterative interactions, reminiscent of a brainstorming session among human experts, they collaboratively craft applications that are not only functional but finely tuned to user requirements [30].

One specific method that can be implemented within such a system is the Soft Prompt Encoder (SPE) [31], in which strategically interspersed prompts are placed within a multi-output generative AI pipeline to enhance the standardization and accuracy of outputs. A similar application is the Synthesizing, Executing, and Debugging (SED) [32] method. It generates draft code, rigorously tests it against predefined cases, and subsequently debugs and refines the code, all within an integrated and automated framework.

4.3. Prioritizing User Interaction in AI Systems



The successful and optimal utilization of generative AI systems relies significantly on intuitive and user-centric interfaces. Systems should be developed to guide users through the intricacies of AI interactions seamlessly. Central to this user experience are design principles that facilitate prompt formulation, combination, application, and representation in user interfaces. An example is PromptIDE, an advanced Integrated Development Environment (IDE) which enables users to experiment with prompt variations, visualize prompt performance, and iteratively optimize prompts [33].

4.4. Autonomous Learning Systems with Human Oversight

An AI system can be developed to continually learn from its interactions with humans and identify improvement opportunities based on these interactions. Human reviewers can then assess the viability and appropriateness of the recommended improvements and prioritize their execution. This synergy of autonomous learning and human oversight holds the potential for a harmonious blend of machine efficiency and human wisdom. This partnership is one where AI systems continuously evolve but always under human experts' watchful and strategic guidance, ensuring a balance between innovation and responsibility.

Developing AI systems comprising multiple embedded generative AI models is the most sophisticated application of generative AI to date. While such systems are optional for engineering teams to reap the benefits of generative AI, it is still important for engineering leaders to be aware of such systems and what is possible in the new era of AI-assisted engineering.

Summary

Generative AI is not merely an evolution in software engineering; it's a revolution. As we stand at the cusp of this transformative era, the software engineering landscape is poised for unprecedented change. The capabilities of generative AI models, from code generation to software design, promise efficiencies and innovations that were once the stuff of science fiction. Yet, with every revolution comes uncertainty.

How will these models integrate with existing systems? What is the proper balance between human expertise and AI autonomy? And critically, how do engineering leaders navigate this new world of AI-assisted engineering?

At Codility, we understand the nuances, opportunities, and challenges that this generative AI revolution presents. We're not just observers; we're active participants dedicated to providing engineering leaders with the insights and tools they need to empower their teams to harness the power of generative AI. Our commitment is to ensure leaders are not left grappling in the dark as the software engineering domain undergoes this seismic shift. Instead, with Codility by their side, they are empowered, informed, and ready to lead their teams into the future.

A critical next step for engineering leaders is to understand the skills software engineers need in this new era of human-AI collaboration and, in turn, how to assess and develop these skills. From there, you will be well on your way to developing an industry-leading AI-powered engineering team.

Book a demo with us to check out Codility's first-of-its-kind assessment of AI-assisted engineering skills. Current customers can contact their Codility representative to learn more about these new assessment offerings today.

Sincere thanks are extended to Ilya Sakharov whose valuable insights and contributions were instrumental in shaping this paper.

[1] Lo, L. S. (2023). The Art and Science of Prompt Engineering: A New Literacy in the Information Age. *Internet Reference Services Quarterly*, 1-8.

[2] Giray, L. (2023). Prompt Engineering with ChatGPT: A Guide for Academic Writers. *Annals of Biomedical Engineering*, 1-5.

[3] Lee, J. S., Kim, J., & Kim, P. M. (2023). Score-based generative modeling for de novo protein design. *Nature Computational Science*, 1-11.

[4] Liu, X., Li, H., & Zhu, X. (2023). A GPT-based method of Automated Compliance Checking through prompt engineering.

[5] Hicks, L. (2023). Louder than a Whisper: Praise for AI Coding Assistants That Doesn't Overpromise. *Medium*.
https://medium.com/slalom-technology/louder-than-a-whisper-praise-for-ai-coding-assistants-that-doesnt-overpromise-9912c9e61db1

[6] Ekedahl, H., & Helander, V. (2023). *Can artificial intelligence replace humans in programming?*

[7] Nikolaidis, N., Flamos, K., Feitosa, D., Chatzigeorgiou, A., & Ampatzoglou, A. The End of an Era: Can Ai Subsume Software Developers? Evaluating Chatgpt and Copilot Capabilities Using Leetcode Problems. *Evaluating Chatgpt and Copilot Capabilities Using Leetcode Problems*.

[8] Maeda, J., & Bolaños, M. (2023). What are Prompts? *Microsoft*.
https://learn.microsoft.com/en-us/semantic-kernel/prompt-engineering/

[9] Ramel, D. (2021). GitHub Copilot Security Study: 'Developers Should Remain Awake' in View of 40% Bad Code Rate. *visualstudiomagazine*.
https://visualstudiomagazine.com/articles/2021/08/26/github-copilot-security.aspx

[10] Dang, H., Mecke, L., Lehmann, F., Goller, S., & Buschek, D. (2022). How to prompt? Opportunities and challenges of zero-and few-shot learning for human-AI interaction in creative applications of generative models. *arXiv preprint arXiv:2209.01390.*

[11] Ahmed, T., Pai, K. S., Devanbu, P., & Barr, E. T. (2023). Improving Few-Shot Prompts with Relevant Static Analysis Products. *arXiv preprint arXiv:2304.06815.*

[12] Reynolds, L., & McDonell, K. (2021, May). 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* (pp. 1-7).

[13] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Quoc, V., & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems, 35*, 24824-24837.

[14] Zhou, Y., Muresanu, A. I., Han, Z., Paster, K., Pitis, S., Chan, H., & Ba, J. (2022). Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910.*

[15] Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., & Narasimhan, K. (2023). Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601.*

[16] Liu, C., Bao, X., Zhang, H., Zhang, N., Hu, H., Zhang, X., & Yan, M. (2023). Improving ChatGPT Prompt for Code Generation. *arXiv preprint arXiv:2305.08360.*

[17] Dang, H., Mecke, L., Lehmann, F., Goller, S., & Buschek, D. (2022). How to prompt? Opportunities and challenges of zero-and few-shot learning for human-AI interaction in creative applications of generative models. *arXiv preprint arXiv:2209.01390.*

[18] Zamfirescu-Pereira, J. D., Wong, R. Y., Hartmann, B., & Yang, Q. (2023, April). Why Johnny can't 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* (pp. 1-21).

[19] Tian, H., Lu, W., Li, T. O., Tang, X., Cheung, S. C., Klein, J., & Bissyandé, T. F. (2023). Is ChatGPT the Ultimate Programming Assistant--How far is it?. *arXiv preprint arXiv:2304.11938.*

[20] Lo, L. S. (2023). The Art and Science of Prompt Engineering: A New Literacy in the Information Age. *Internet Reference Services Quarterly*, 1-8.

[21] Liu, C., Bao, X., Zhang, H., Zhang, N., Hu, H., Zhang, X., & Yan, M. (2023). Improving ChatGPT Prompt for Code Generation. *arXiv preprint arXiv:2305.08360.*

[22] Mao, J., Middleton, S. E., & Niranjan, M. (2023). Prompt position really matters in few-shot and zero-shot NLU tasks. *arXiv preprint arXiv:2305.14493.*

[23] Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2023). Lost in the Middle: How Language Models Use Long Contexts. *arXiv preprint arXiv:2307.03172.*

[24] Wang, Z. J., Montoya, E., Munechika, D., Yang, H., Hoover, B., & Chau, D. H. (2022). Diffusiondb: A large-scale prompt gallery dataset for text-to-image generative models. *arXiv preprint arXiv:2210.14896.*

[25] Acar, O. A. (2023). AI Prompt Engineering Isn't the Future. *Harvard Business Review.*

[26] Chen, L., Zaharia, M., & Zou, J. (2023). How is ChatGPT's behavior changing over time?. *arXiv preprint arXiv:2307.09009.*

[27] Dodge, J., Ilharco, G., Schwartz, R., Farhadi, A., Hajishirzi, H., & Smith, N. (2020). Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305.*

[28] Ding, N., Qin, Y., Yang, G., Wei, F., Yang, Z., Su, Y., Hu, S., Chen, Y., Chen, W., & Sun, M. (2023). Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence, 5*(3), 220-235.

[29] Weisz, J. D., Muller, M., He, J., & Houde, S. (2023). Toward general design principles for generative AI applications. *arXiv preprint arXiv:2301.05578.*

[30] Li, G., Hammoud, H. A. A. K., Itani, H., Khizbullin, D., & Ghanem, B. (2023). Camel: Communicative agents for "mind" exploration of large-scale language model society. *arXiv preprint arXiv:2303.17760.*

[31] Chuang, Y. N., Tang, R., Jiang, X., & Hu, X. (2023). Spec: A soft prompt-based calibration on mitigating performance variability in clinical notes summarization. *arXiv preprint arXiv:2303.13035.*

[32] Liventsev, V., Grishina, A., Härmä, A., & Moonen, L. (2023). Fully Autonomous Programming with Large Language Models. *arXiv preprint arXiv:2304.10423.*

[33] Strobelt, H., Webson, A., Sanh, V., Hoover, B., Beyer, J., Pfister, H., & Rush, A. M. (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.