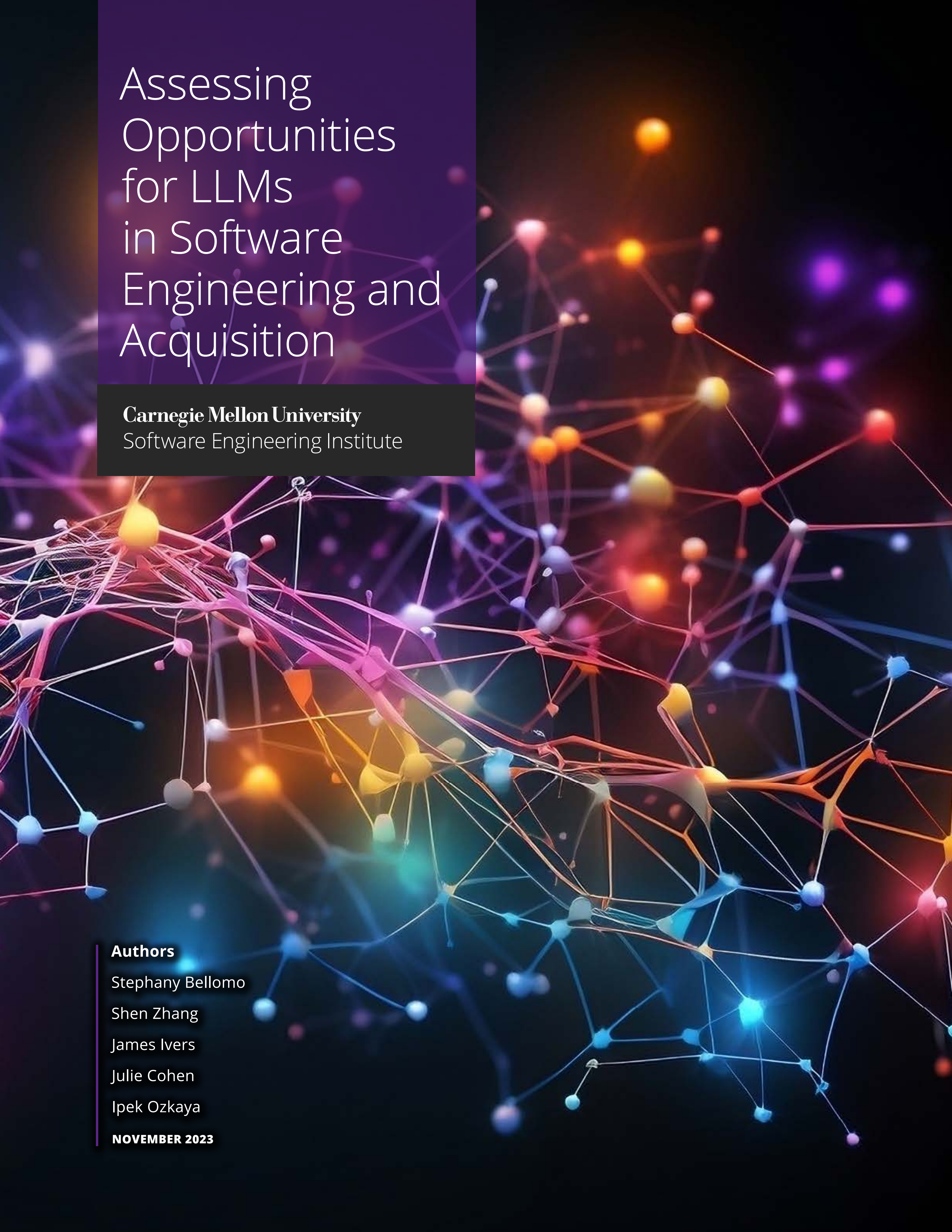


Assessing Opportunities for LLMs in Software Engineering and Acquisition

Carnegie Mellon University
Software Engineering Institute

Authors

Stephany Bellomo

Shen Zhang

James Ivers

Julie Cohen

Ipek Ozkaya

NOVEMBER 2023

LARGE LANGUAGE MODELS (LLMS) ARE GENERATIVE ARTIFICIAL INTELLIGENCE (AI) MODELS that have been trained on massive corpuses of text data and can be prompted to generate new, plausible content. LLMs are seeing rapid advances, and they promise to improve productivity in many fields. OpenAI's GPT-4[1] and Google's LaMDA[2] are the underlying LLMs of services like ChatGPT[3], CoPilot[4], and Bard[5]. These services can perform a range of tasks, including generating human-like text responses to questions, summarizing artifacts, and generating working code. These models and services are the focus of extensive research efforts across industry, government, and academia to improve their capabilities and relevance, and organizations in many domains are rigorously exploring their use to uncover potential applications.

The idea of harnessing LLMs to enhance the efficiency of software engineering and acquisition activities holds special allure for organizations with large software operations, such as the Department of Defense (DoD), as doing so offers the promise of substantial resource optimization. Potential use cases for LLMs are plentiful, but knowing how to assess the benefits and risks associated with their use is nontrivial. Notably, to gain access to the latest advances, organizations may need to share proprietary data (e.g., source code) with service providers. Understanding such implications is central to intentional and responsible use of LLMs, especially for organizations managing sensitive information.

In this document, we examine how decision makers, such as technical leads and program managers, can assess the fitness of LLMs to address software engineering and acquisition needs [Ozkaya 2023]. We first introduce exemplar scenarios in software engineering and software acquisition and identify common archetypes. We describe common concerns involving the use of LLMs and enumerate tactics for mitigating those concerns. Using these common concerns and tactics, we demonstrate how decision makers can assess the fitness of LLMs for their own use cases through two examples.

Capabilities of LLMs, risks concerning their use, and our collective understanding of emerging services and models are evolving rapidly [Brundage et al. 2022]. While this document is not meant to be comprehensive in covering all software engineering and acquisition use cases, their concerns, and mitigation tactics, it demonstrates an approach that decision makers can use to think through their own LLM use cases as this space evolves.

What Is an LLM?

An LLM is a deep neural network model trained on an extensive corpus of diverse documents (e.g., websites and books) to learn language patterns, grammar rules, facts and even some reasoning abilities [Wolfram 2023]. LLMs can generate responses to inputs ("prompts") by iteratively determining the next word or phrase appearing after others based on the prompt and patterns and associations learned from their training corpus using probabilistic and randomized selection [White et al. 2023]. This capability allows LLMs to generate human-like text that can be surprisingly coherent and contextually relevant, even if they may not always be semantically correct.

While LLMs can perform complex tasks using their trained knowledge, they lack true understanding. Rather, they are sophisticated pattern matching tools. Moreover, due to their probabilistic reasoning, they can generate inaccurate results (often referred to as "hallucinations"), such as citations to non-existent references or method calls to nonexistent application programming interfaces (APIs). While LLMs can perform analysis and inferencing on new data they have been prompted with, data on which LLMs have been trained can limit their accuracy. However, the technology is rapidly advancing with new models having increasing complexity and parameters, and benchmarks have already emerged for comparing their performance [lmsys 2023]. In addition, LLM service providers are working on ways to use more recent data [D'Cruze 2023]. Despite these limitations, there are productive uses of LLMs today.

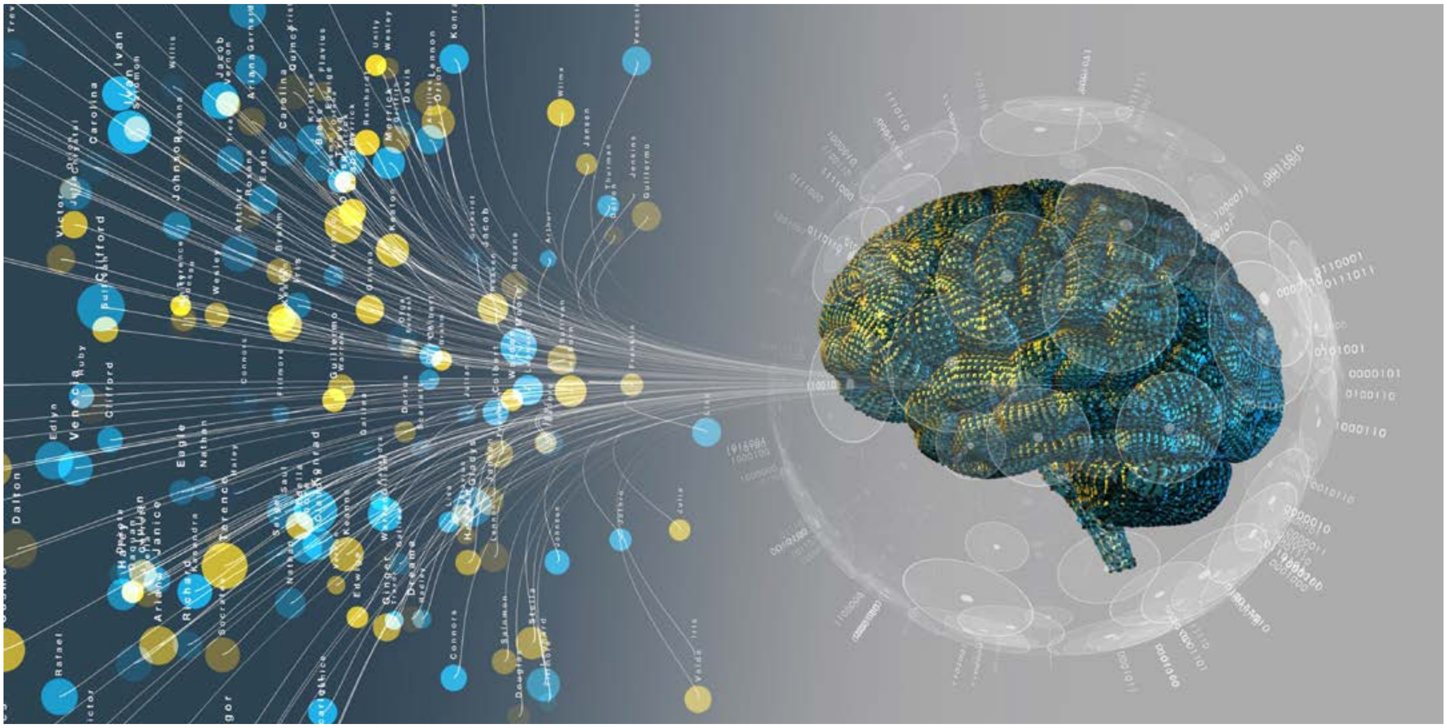
Choosing an LLM

There are already dozens of LLMs and services built using LLMs, and more emerge every day. These models vary in many dimensions, from technical to contractual, and the details of these differences can be difficult to keep straight. The following distinctions are a good starting point when choosing an LLM for use.

Model or Service. ChatGPT is a chatbot built on OpenAI's GPT family of LLMs [Open AI 2023]. The difference is important, as services built on LLMs can add additional capabilities (e.g., specialized chatbot features, specialized training beyond the core LLM, or non-LLM features that can improve results from an LLM). A *service* like ChatGPT is typically hosted by a service provider, meaning that it manages the computing resources (and associated costs) and that users are typically required to send their prompts (and potentially sensitive data) to the service provider to use the service. A *model,* like Meta's Llama 2[6], can be fine-tuned with domain- or organization-specific data to improve accuracy, but it typically lacks the added features and resources of a commercially supported service.

[1] **https://openai.com/research/gpt-4**
[2] **https://blog.google/technology/ai/lamda/**
[3] **https://chat.openai.com**
[4] **https://github.com/features/copilot**
[5] **https://bard.google.com**

[6] **https://ai.meta.com/llama/**



General or Specialized. LLMs are pre-trained on a corpus, and the composition of that corpus is a significant factor affecting an LLM's performance. General LLMs are trained on text sources like Wikipedia that are available to the public. Specialized LLMs fine tune those models by adding training material from specific domains like healthcare and finance [Zhou et al. 2022; Wu et al. 2023]. LLMs like CodeGen[7] have been specialized with large corpuses of source code for use in software engineering.

Open Source or Proprietary. *Open source LLMs* provide a platform for researchers and developers to freely access, use, and even contribute to the model's development. *Proprietary LLMs* are subject to varying restrictions on use, making them less open to experimentation or potential deployment. Some providers (e.g., Meta) use a license that is largely, but not completely, open [Hull 2023]. OpenAI offers a different compromise: While the GPT series of LLMs is not open source, OpenAI does permit fine tuning (for a fee) as a means of specialization and limited experimentation with their proprietary model.

> The field of LLMs is a fast-moving space. Moreover, the ethics and regulations surrounding their use are also in a state of flux, as society grapples with the challenges and opportunities these powerful models present. Keeping apprised of these developments is crucial for taking advantage of the potential offered by LLMs.

Use Cases

The ability of LLMs to generate plausible content for text and code applications has sparked the imaginations of many. A recent literature review examines 229 research papers written since 2017 on the application of LLMs to software engineering problems [Hou et al. 2023]. Application areas span requirements, design, development, testing, maintenance, and management activities, with development and testing being the most common.

Our team, which works with government organizations daily, took a broader perspective and brainstormed several dozen ideas for using LLMs in common software engineering and acquisition activities (see Table 1 for examples). Two important observations quickly emerged from this activity. First, most use cases represent human-AI partnerships in which an LLM or LLM-based service could be used to help humans (as opposed to replace humans) complete tasks more quickly. Second, deciding which use cases would be most feasible, beneficial, or affordable is not a trivial decision for those just getting started with LLMs.

[7] **https://github.com/salesforce/CodeGen**

Table 1: Sample Acquisition and Software Engineering Use Cases

ACQUISITION USE CASES	SOFTWARE ENGINEERING USE CASES
A1. A new acquisition specialist uses an LLM to generate an overview of relevant federal regulations for an upcoming request for proposal (RFP) review, expecting the summary to save time in background reading.	SE1. A developer uses an LLM to find vulnerabilities in existing code, hoping that the exercise will catch additional issues not already found by static analysis tools.
A2. A chief engineer uses an LLM to generate a comparison of alternatives from multiple proposals, expecting it to use the budget and schedule formulas from previous similar proposal reviews and generate accurate itemized comparisons.	SE2. A developer uses an LLM to generate code that parses structured input files and performs specified numerical analysis on its inputs, expecting it to generate code with the desired capabilities.
A3. A contract specialist uses an LLM to generate ideas for a request for information (RFI) solicitation given a set of concerns and vague problem description, expecting it to generate a draft RFI that is at least 75% aligned with their needs.	SE3. A tester uses an LLM to create functional test cases, expecting it to produce a set of text test cases from a provided requirements document.
A4. A CTO uses an LLM to create a report summarizing all uses of digital engineering technologies in the organization based on internal documents, expecting it can quickly produce a clear summary that is at least 90% correct.	SE4. A developer uses an LLM to generate software documentation from code to be maintained, expecting it to summarize its functionality and interface.
A5. A program office lead uses an LLM to evaluate a contractor's code delivery for compliance with required design patterns, expecting that it will identify any instances in which the code fails to use required patterns.	SE5. A software engineer who is unfamiliar with SQL uses an LLM to generate an SQL query from a natural language description, expecting it to generate a correct query that can be tested immediately.
A6. A program manager uses an LLM to summarize a set of historical artifacts from the past six months in preparation for a high-visibility program review and provides specific retrieval criteria (e.g., delivery tempo, status of open defects, and schedule), expecting it to generate an accurate summary of program status that complies with the retrieval criteria.	SE6. A software architect uses an LLM to validate whether code that is ready for deployment is consistent with the system's architecture, expecting that it will reliably catch deviations from the intended architecture.
A7. A program manager uses an LLM to generate a revised draft of a statement of work, given a short starting description and a list of concerns (e.g., cybersecurity, software delivery tempo, and interoperability goals). The program manager expects it to generate a structure that can be quickly refined and that includes topics drawn from best practices they may not think to request explicitly.	SE7. A developer uses an LLM to translate several classes from C++ to Rust, expecting that the translated code will pass the same tests and be more secure and memory safe.
A8. A requirements engineer uses an LLM to generate draft requirements statements for a program upgrade based on past similar capabilities, expecting them to be a good starting point.	SE8. A developer uses an LLM to generate synthetic test data for a new feature being developed, expecting that it will quickly generate syntactically correct and representative data.
A9. A contract officer is seeking funding to conduct research on a high-priority topic they are not familiar with. The contract officer uses an LLM to create example project descriptions for their context, expecting it to produce reasonable descriptions.	SE9. A developer provides an LLM with code that is failing in production and a description of the failures, expecting it to help the developer diagnose the root cause and propose a fix.

Archetypes

Commonalities among the use cases lend themselves to abstracting the set into a manageable number of archetypes. Two dimensions are helpful in this regard: the nature of the activity an LLM is performing and the nature of the data that the LLM is acting on. Taking the cross-product of these dimensions, these use cases fall into the archetypes depicted in Table 2.

Table 2: Use Case Archetypes

ACTIVITY TYPE	DATA TYPE			
	Text	Code	Model	Images
Retrieve Information	retrieve-text	retrieve-code	retrieve-model	retrieve-images
Generate Artifact	generate-text	generate-code	generate-model	generate-images
Modify Artifact	modify-text	modify-code	modify-model	modify-images
Analyze Artifact	analyze-text	analyze-code	analyze-model	analyze-images

Matching a specific use to an archetype helps identify common concerns among similar use cases and known solutions commonly applied for similar use cases. Archetypes can be a tool that organizations use to group successes, gaps, and lessons learned in a structured way.

Activity Type captures differences in associations that an LLM would need to make to support a use case, with some asking an LLM to do things that a language model was not designed to do:

- *Retrieve Information* asks an LLM to construct a response to a question (e.g., what's the Observer pattern?) for which a known answer is likely found in the training corpus, directly or across related elements.
- *Generate Artifact* asks an LLM to create a new artifact (e.g., a summary of a topic or a Python script that performs a statistical analysis) that likely bears similarity with existing examples in the corpus.
- *Modify Artifact* asks an LLM to modify an existing artifact to improve it in some way (e.g., translate Python code to Java or remove a described bug) that resembles analogous improvements among artifacts in the training corpus.
- *Analyze Artifact* asks an LLM to draw a conclusion about provided information (e.g., what vulnerabilities are in this code or will this architecture scale adequately?) that likely requires semantic reasoning about data.

Data Type captures differences in the kind of data that an LLM operates on or generates, such as the differences in semantic rules that make data (e.g., code) well-formed:

- *Text* inputs vary widely in formality and structure (e.g., informal chat versus structured text captured in templates).
- *Code* is text with formal rules for structure and semantics, and a growing number of LLMs are being specialized to take advantage of this structure and semantics.
- *Models* are abstractions (e.g., from software design or architecture) that often use simple terms (e.g., publisher) that imply deep semantics.
- *Images* are used to communicate many software artifacts (e.g., class diagrams) and often employ visual conventions that, much like models, imply specific semantics. While LLMs operate on text, multimodal LLMs (e.g., GPT-4) are growing in their ability to ingest and generate image data.

Figure 1 shows an example of using the archetypes to generate ideas for LLM use cases in a particular domain. This example focuses on independent verification and validation (IV&V), a resource-intensive activity within the DoD that involves many different activities that might benefit from the use of LLMs. More complex use cases for IV&V could also be generated that involve integration of multiple archetypes into a larger workflow.

ACTIVITY TYPE	DATA TYPE			
	Text	Code	Model	Images
Retrieve Information	retrieve-text	retrieve-code	retrieve-model	retrieve-images
Generate Artifact	generate-text ❶ ❷	generate-code ❹	generate-model	generate-images ❻
Modify Artifact	modify-text	modify-code	modify-model	modify-images
Analyze Artifact	analyze-text ❸	analyze-code ❺	analyze-model	analyze-images

❶ A tester uses an LLM to create integration test descriptions from a set of APIs and integration scenarios, expecting it to produce a set of test case descriptions that can be used to implement tests.

❸ An IV&V evaluator uses an LLM to analyze software design documents against a specific set of certification criteria and to generate a certification report, expecting it to describe certification violations that they will review to confirm.

❺ A developer uses an LLM to find vulnerabilities in existing code, hoping that the exercise will catch additional issues not already found by static analysis tools.

❷ An IV&V evaluator uses an LLM to create a verification checklist from a set of certification regulations and a system description, expecting it to produce a context-sensitive checklist they can tailor.

❹ A new developer uses an LLM as a pair programmer to write code, expecting it to help create vulnerability-free code.

❻ A developer uses an LLM to create a network view for authorization to operate (ATO) certification from a description of the architecture, expecting it to produce a rough network diagram they can refine.

Figure 1: Using Archetypes to Help Brainstorm Potential Use Cases

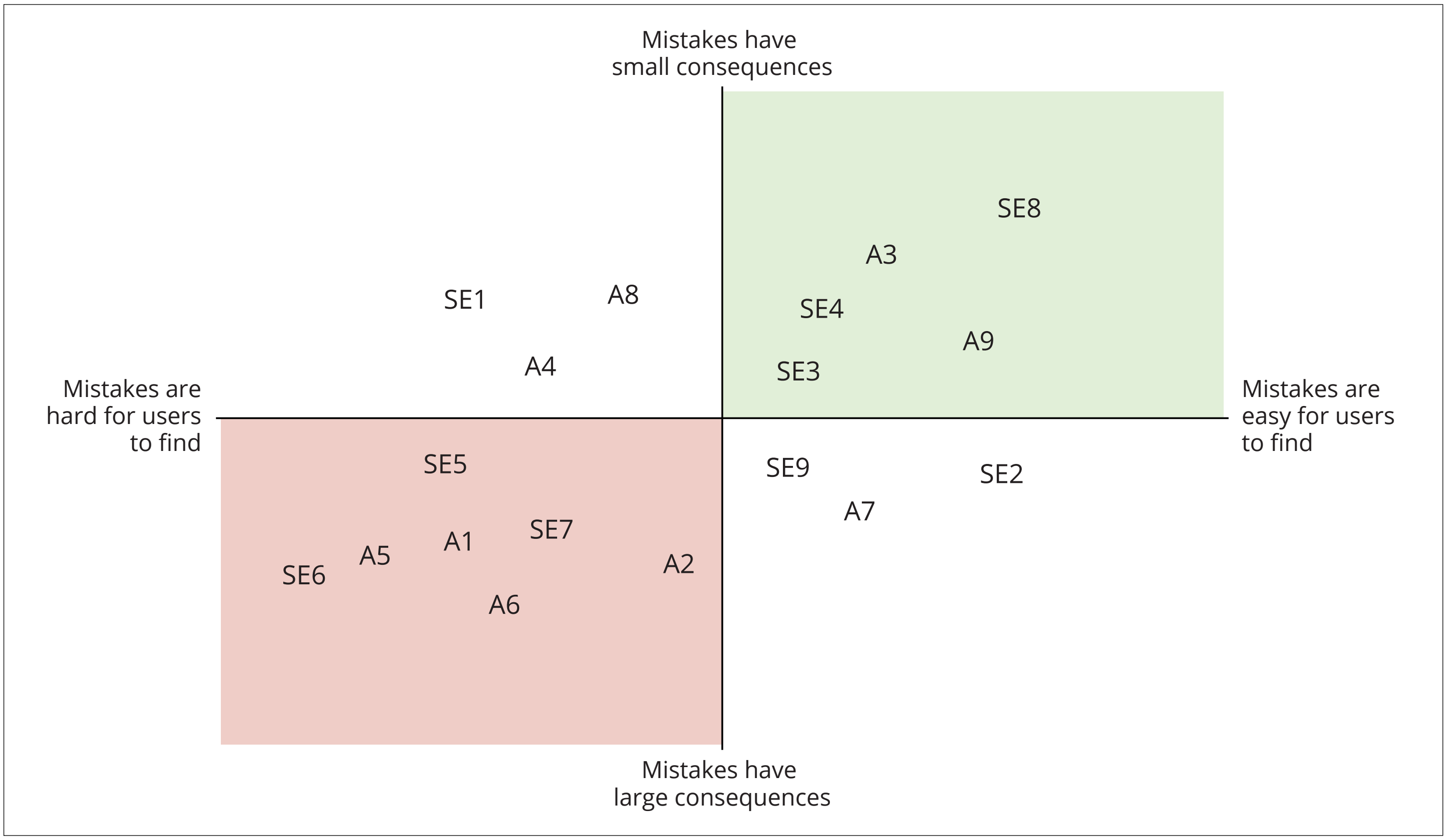

Mistakes have small consequences
Mistakes are hard for users to find
Mistakes are easy for users to find
Mistakes have large consequences
SE8
A3
SE1
A8
SE4
A9
A4
SE3
SE5
SE9
SE2
A7
SE7
A1
A5
A2
SE6
A6

Figure 2: Two Ways to Look at Concerns with the Generation of Incorrect Results (A: Acquisition Use Cases, SE: Software Engineering Use Cases [Table 1])

Concerns and How to Address Them

Recognizing **concerns** around applications of LLMs to software engineering and acquisition, and deciding how to address each, will help decision makers make more informed choices. There are multiple perspectives one should consider before going forward with an LLM use case. An important reality is that the results generated by LLMs are in fact sometimes wrong. Figure 2 illustrates this perspective based on two questions:

- How significant would it be to act on an incorrect result in a given use case?
- How easy would it be for a user in the use case to recognize that a result from an LLM is incorrect?

This figure shows a notional placement of the use cases from Table 1 (actual placement would be reliant on refinement of these use cases). The green quadrant is ideal from this perspective: Mistakes are not particularly consequential and relatively easy to spot. Use cases in this quadrant can be a great place for organizations to start LLM experimentation. The red quadrant, on the other hand, represents the least favorable cases for LLM use: Mistakes create real problems and are hard for users to recognize.

The consequences of mistakes and ease of spotting them is only one perspective of evaluation. Another perspective is the expected significance of improvements or efficiencies achievable with LLMs. Among many concerns, we discuss five categories in further detail in this document—correctness, disclosure, usability, performance, and trust—as they are relevant to all use cases.

Correctness: The significance of correctness as a concern depends on factors such as how the results will be used, the safeguards used in workflows, and the expertise of users. Correctness refers to *the overall accuracy and precision of output relative to some known truth or expectation.* Accuracy hinges greatly on whether an LLM was trained or fine-tuned with data that is sufficiently representative to support the specific use case. Even with rich training corpuses, some inaccuracy can be expected [Ouyang et al. 2023]. For example, a recent study on code translation found GPT-4 to perform better than other LLMs, even though more than 80% of translations on a pair of open source projects contained some errors. Advances are likely to improve, but not eliminate, these numbers [Pan et al. 2023].

Disclosure: When users interact with LLMs, some use cases may require disclosing proprietary or sensitive information to a service provider to complete a task (e.g., sharing source code to help debug it). The disclosure concern is therefore related to *the amount of proprietary information that must be exposed during use.* If users share confidential data, trade secrets, or personal information, there is a risk that such data could be stored, misused, or accessed by unauthorized individuals. Moreover, it might become part of the training data corpus and disseminated without users having any means to track its origin. For example, GSA CIO IL-23-01 (the U.S. General Services Administration instructional letter *Security Policy for Generative Artificial Intelligence [AI] Large Language Models [LLMs]*) bans disclosure of federal nonpublic information as inputs in prompts to third-party LLM endpoints [GSA 2023].

Usability: LLM users have vastly different backgrounds, expectations, and technical abilities. Usability captures the *ability of LLM users with different expertise to complete tasks.* Users may need expertise on both the input (crafting appropriate prompts) and output (judging the correctness of results) sides of LLM use [Zamfirescu-Pereira et al. 2023]. The significance of usability as a concern depends on the degree to which getting to acceptable results is sensitive to the expertise of users. A study completed with developers' early experiences using CoPilot reflects that there is a shift from writing code to understanding code when using LLMs on coding tasks [Bird et al. 2023]. This observation hints at the need for different usability techniques for interaction mechanisms, as well as the need to account for expertise.

Performance: While using an LLM requires much less computing power than training an LLM, responsiveness can still be a factor in LLM use, especially if sophisticated prompting approaches are incorporated into an LLM-based service. For the purposes of concerns related to use cases, performance expresses *the time required to arrive at an appropriate response.* Model size, underlying compute power, and where the model runs and is accessed from are among the factors that influence responsiveness [Patterson et al. 2022]. Services built on LLMs may introduce additional performance overhead due to the way in which other capabilities are integrated with the LLM.

Trust: To employ the technology with the requisite level of trust, users must grasp the limitations of LLMs. Trust reflects *the user's confidence in the output.* Overreliance on an LLM without understanding its potential for error or bias can lead to undesirable consequences [Rastogi et al. 2023]. As a result, several other concerns (e.g., explainability, bias, privacy, security, and ethics) are often considered in relationship to trust [Schwartz et al. 2023]. For example, the DoD published ethical AI principles to advance trustworthy AI systems [DoD 2020].

How significant these and other concerns are for each use case will vary by context and use. The questions provided in Table 3 can help organizations assess how relevant each concern is for a specific use case. A starting point could be to categorize the significance of each concern as High, Medium, or Low. This information can help organizations decide whether an LLM is fit for purpose and what concerns need to be mitigated to avoid unacceptable outcomes.

Table 3: Example Questions to Help Determine the Significance of Common Concerns for a Specific Use Case

CONCERN	SIGNIFICANCE QUESTIONS
Correctness	• What is the risk or impact of using an incorrect result in the use case? • How difficult is it for the expected user to determine whether a result is correct? • Are there gaps in the data used to train the LLM that could adversely impact results (e.g., the data is not current with recent technology releases or contains little data for an esoteric programming language)?
Disclosure	• Can an LLM be prompted without disclosing proprietary information (e.g., using generic questions or abstracting proprietary details)? • What is the risk or impact of a third party being able to observe your prompts? • Are there existing data disclosure constraints that strictly need to be observed?
Usability	• How adept are expected users at prompting an LLM? • How familiar are expected users with approaches for determining whether results are inaccurate? • How familiar are expected users with approaches for determining whether results are incomplete?
Performance	• How quickly must a user or machine be able to act on a result? • Are there significant computing resource limitations? • Are there intermediate steps in the interaction with the LLM that may affect end-to-end performance?
Trust	• Are your expected users predisposed to accept generated results (automation bias) or reject them? • Is the data the LLM was trained on free of bias and ethical concerns? • Has the LLM been trained on data that is appropriate for use?

These common concerns, and questions to determine their significance, enable identification of common **tactics** for addressing each concern. A tactic is a course of action that can be taken to reduce the occurrence or impact of a concern. Table 4 summarizes a collection of tactics that can help mitigate each concern, along with a rough estimate (High [H], Medium [M], or Low [L]) of the relative potential cost of using each tactic. Typically, the more resources (human and computation) a tactic requires, the higher the cost. For example, prompt engineering and model training both address correctness, but prompt engineering is typically much less expensive. Of note, some tactics (purple rows) focus on technical interventions, others (green rows) focus on human-centered actions, and the rest (gray rows) could employ technical or human-centered interventions.

Table 4: Tactics That Can Be Used to Address Common Concerns with LLM Use

CONCERN	TACTIC	DESCRIPTION	COST
Correctness	Prompt engineering	Educate users on prompt engineering techniques and patterns to generate better results.	L
	Validate manually	Dedicate time to allow users to carefully validate interim and final results.	M
	Adjust settings	Change settings of exposed model parameters like temperature (randomness of the model's output) and the maximum number of tokens.	L
	Adopt newer model	Use newer models that integrate technical advances or improved training corpuses that can produce better results.	M
	Fine tune model	Tailor a pretrained model using organization- or domain-specific data to improve results.	M
	Train new model	Use a custom training corpus or proprietary data to train a new model.	H
Disclosure	Open disclosure policy	Establish a policy that allows users to share as much detail as needed to complete a task.	L
	Self-redact sensitive information	Limit what information users can share by establishing guidelines and safeguards, such as using anonymization or limiting requests to generic topics.	M
	Use a local LLM	Deploy an LLM that runs on local hardware and does not share any information with a service provider, enabling disclosure in a trusted context.	H
Usability	Prompt engineering	Educate users on prompt engineering techniques and patterns to generate better results.	L
	Define evaluation criteria a priori	Provide enforceable criteria for the model's output to be checked against before the output is known.	M
Performance	Use more efficient model	Select a model that generates comparable results more efficiently (e.g., perhaps based on new, more advanced algorithms).	L
	Use less capable model	Select a model that generates lower quality results more efficiently.	L
	Increase resources	Increase compute resources or pay for a higher service tier.	M
Trust	Educate users	Educate users on the limitations of LLMs and how to validate the output.	M
	Compare output with non-LLM sources	Validate LLM output against similar information from non-LLM sources, such as the original documents, the internet, or SMEs.	H
	Compare output with other LLMs	Validate LLM output against other trusted LLMs.	H

There is no uniform, context-free answer for addressing any of the concerns discussed above. Each tactic involves different strategies, some of which require changing user behavior, while others require changing the compute environment, data, or the model. Each tactic impacts costs in different ways. Moreover, a single tactic may sometimes help address multiple concerns.

Just as the capabilities of LLMs will continue to evolve, the significance of different concerns for an organization's use cases may also change. Assess these and other tactics in the context of your concerns and use cases, and choose wisely to meet your needs.

Deciding Whether LLMs Are a Fit

Analysis of how the concerns would be expected to apply across the use case archetypes (Table 2) helps identify higher- and lower-risk applications of LLMs in software engineering and acquisition. For example

- Concerns about disclosing sensitive information span all activity types but is least common (and more manageable) in *Information Retrieval* use cases.
- Correctness of LLM results is a significant concern across all activity types. However, it is most significant for activities that generate outputs that are difficult to validate, such as in *Analyze Artifact* use cases.
- *Analyze Artifact* use cases involving unstructured text are likely to pose greater challenges, given the variation in a training corpus (e.g., reviewing RFPs for completeness against criteria).

Such insights are helpful but limited. How does an organization gain greater insight into whether an LLM is a good fit for a particular use case? The concerns and tactics described in this document should be used in conjunction with existing system and software evaluation and risk mitigation practices to assess specific needs and options. Common sense applies, of course. The rigor with which an organization deliberates on the use of a new technology should be proportional with the cost of experiments. For example, if a developer wants to spend an afternoon experimenting with CoPilot, there's no need for an extensive analysis of alternatives. As the investment goes up (e.g., basing a six-month effort on LLMs), so should the due diligence. The following steps suggest one possible way to approach assessing the suitability of LLMs for your intended use.

Clarify your use case. Once you have a use case to pursue, it's important to get specific about the scope of the use case (e.g., what begins and ends it) and how LLMs will be used. Characterize what success looks like (e.g., target measures for improvement over existing workflows). Characterize the familiarity with LLMs and domain expertise of expected users.

Identify your concerns. In addition, we defined five broad categories of *concerns* that often matter when using LLMs. How many are relevant to your use case? Which are more important than others? Consider a simple High/Medium/Low ranking for each concern (e.g., using the questions in Table 3 as a starting point). Those concerns that matter most will drive your design and implementation decisions, such as which tactics to employ.

Design your workflow. A new workflow to realize your use case requires specifying what input is provided to LLMs, what output is expected, what role users will play throughout the process, and what tactics will be used to address your concerns. Tactics vary in effectiveness and cost, among other factors, and there is no single tactic that is the right answer for all use cases. Designing an appropriate workflow often requires use of multiple tactics and assessing their tradeoffs based on your available resources.

Assess suitability for your use case. Look at the combination of tactics that you plan to use and start with simple qualitative analysis. The following considerations are likely to be helpful.

- How effective are the selected tactics in addressing your most significant concerns? Concerns that are more important than others for your use case often require more expensive mitigations or impose constraints on your solution.
- Are all the tactics necessary to address your concerns? For example, good practices, such as manually validating results, may be overkill for use cases like using LLMs to aid brainstorming.
- What is the estimated cost of all tactics and does the use case warrant that investment? For example, highly effective tactics, such as training a new model on your own data, can also require a significant investment.

We illustrate the use of these steps to guide decisions for a software engineering use case in Table 5 and for an acquisition use case in Table 6. While both examples have a high *Disclosure* concern, they employ different tactics.

Table 5: Illustration of Assessing the Use of LLMs for a Software Engineering Use Case

Idea

A developer wants to use an LLM to write a new user interface (UI) for an application, using an unfamiliar framework (Angular) that other teams are using.

Clarify Use Case

The developer knows the domain and application but is new to using LLMs. The developer expects to describe the needed features to an LLM and requests code to be generated that mostly gets the job done. The developer expects to review the code and manually fix anything that does not work correctly.

Identify Concerns

Correctness and *Trust* are low-priority concerns, as the developer plans to test code in small increments as features are added to get rapid feedback. *Disclosure* is a high-priority concern, as the UI supports internal practices that the developer's organization is not comfortable with sharing. *Usability* is a medium-priority concern as the developer is new to LLMs and unsure how to get the best results. *Performance* is a low-priority concern as the use case falls well within the expected response time.

Design Workflow

Disclosure is the driving concern as it was identified as high priority. The developer prefers to use an LLM-based service like ChatGPT, but doing so requires embedding proprietary information in prompts. The developer considers a local LLM, but the work and resources involved to get started in an unfamiliar area is discouraging. The developer confers with management and agrees on a *self-redacting* strategy. The developer will limit prompts to generic questions (e.g., create a five-column table that users can sort based on any column) that avoid information that identifies the organization and meaning of data being displayed. The developer is aware that this approach means the LLM will only be able to generate the scaffolding for the UI and that manual effort will be required to tailor the scaffolding to the application's data.

Assess Suitability

Management is satisfied that the developer can use the *self-redacting* strategy to implement a more modest version of the original use case without disclosing sensitive information. The developer has been reviewing *prompt engineering* material to prepare, easing the *usability* concerns.

Table 6: Illustration of Assessing the Use of LLMs for an Acquisition Use Case

Idea

A government program manager issued an RFI to collect technical challenges to a proposed change and wants to use an LLM to summarize the challenges found in the dozens of responses received.

Clarify Use Case

The RFI responses did not follow a common structure, but they appear rich in information. The manager expects that if their team can provide the responses to an LLM, the LLM can generate useful summaries in a fraction of the time required to read and manually analyze all responses. The manager wants to see how well the LLM can do at summarizing individual responses and collecting trends across the responses. The manager is eager to see if the results are correct, as their team must regularly deal with large numbers of RFI responses and improving efficiency and correctness would have significant benefits.

Identify Concerns

Disclosure is a high-priority concern, as the responses cannot be provided to a third-party service. *Correctness* and *Trust* are high-priority concerns, as the manager wants to avoid summarizations that omit challenges and would like to quickly make reliable decisions. *Usability* and *Performance* are low-priority concerns for now, but they may need to be revisited if use of LLMs for RFI summarization becomes routine in the organization.

Design Workflow

An LLM-based service is available with appropriate approval, and the manager will use this instance. Using a service for which *Disclosure* is already approved mitigates the *Disclosure* concern. The manager is unsure of the best path to feed the data to the LLM-based service; *fine tuning* the model with the RFI responses is one path, but they could also try *embedding the responses in prompts.* The manager decides to test the less expensive option *(prompting)* with one example to get a better handle on the option.

Assess Suitability

The manager is happy with the approved option. The results have gaps, though, likely because the LLM was not trained with similar material. The manager decides to pursue *fine tuning* to improve results in the next iteration. The internal data-science team is tasked with the *fine tuning.*

The application of a growing body of knowledge, like the concerns and tactics described in this document, can help organizations quickly understand the implications of proposed LLM use.

Note, however, that LLM technology is rapidly changing, and the answers to your questions today may not hold tomorrow. Practitioners and researchers will continue to innovate and discover new tactics. Model improvements may reduce the significance of some concerns. The cost of some tactics may dramatically decrease over time, such as fine tuning an LLM or hosting your own local LLM. Pay attention to such improvements and reassess your decisions as needed.

Conclusions

LLMs have the potential to significantly improve broad collections of software engineering and acquisition practices, lowering costs, improving response times, and providing new insights. It's important, however, to acknowledge that adoption of LLMs comes with risks, as described in this document. In some cases, these risks may lead to increased costs due to the tactics that must be employed to ensure acceptable outcomes in different use cases. Assessing applications of LLMs and understanding their implications is key to responsible use of this new technology and making good investment decisions.

This document covers only the most common tactics, and as experience with the use of LLMs grows more tactics are likely to emerge. Evaluating where your specific use cases fall within the intersection of your tolerance for potential errors, your ability to quickly identify those errors, your primary concerns, and the resources available for mitigating risks enables selecting the most appropriate tactics. This assessment serves as a foundational step in determining whether LLMs can deliver a return on investment within your unique context. The strategies and examples provided in this document are helpful starting points.

References

[Bird et al. 2023]
Bird, Christian; Ford, Denae; Zimmermann, Thomas; Forsgren, Nicole; Kalliamvakou, Eirini; Lowdermilk, Travis; and Gazit, Idan. Taking Flight with Copilot. *Communications of the ACM.* Volume 66. Number 6. June 2023. Pages 56–62. **https://cacm.acm.org/magazines/2023/6/273221-taking-flight-with-copilot/abstract**

[Brundage et al. 2022]
Brundage, M.; Mayer, K.; Eloundou, T.; Agarwal, S.; Adler, S.; Krueger, G.; Leike, J.; & Mishkin, P. Lessons Learned on Language Model Safety and Misuse. OpenAI. March 2, 2022. **https://openai.com/research/language-model-safety-and-misuse**

[D'Cruze 2023]
D'Cruze, Danny. ChatGPT gets big update! OpenAI's chatbot will now be able to access real-time data on internet; check how to use it. *Business Today.* September 28, 2023. **https://www.businesstoday.in/technology/news/story/chatgpt-gets-big-update-openais-chatbot-will-now-be-able-to-access-real-time-data-on-internet-check-how-to-use-it-399981-2023-09-28**

[DoD 2020]
U.S. Department of Defense. DOD Adopts Ethical Principles for Artificial Intelligence. *U.S. Department of Defense.* Feb. 24, 2020. **https://www.defense.gov/News/Releases/Release/Article/2091996/dod-adopts-ethical-principles-for-artificial-intelligence/**

[GSA 2023]
General Services Administration. Security Policy for Generative Artificial Intelligence (AI) Large Language Models (LLMs). *U.S. General Services Administration.* June 9, 2023. **https://www.gsa.gov/directives-library/security-policy-for-generative-artificial-intelligence-ai-large-language-models-llms**

[Hou et al. 2023]
Hou, X.; Zhao, Y.; Liu Y.; Yang, Z; Wang, K.; Li, L.; Luo, X.; Lo, D.; Grundy, J.; & Wang, H. Large Language Models for Software Engineering: A Systematic Literature Review. *arXiv.* September 12, 2023. **https://arxiv.org/pdf/2308.10620.pdf**

[Hull 2023]
Hull, Charlie. Is Llama 2 open source? No—and perhaps we need a new definition of open…. *OpenSource Connections.* July 19, 2023. **https://opensourceconnections.com/blog/2023/07/19/is-llama-2-open-source-no-and-perhaps-we-need-a-new-definition-of-open/**

[lmsys 2023]
lmsys. Leaderboard (chatbot-arena-leaderboard). *Huggingface.* October 3, 2023 [accessed]. **https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard**

[Open AI 2023]
OpenAI. GPT-4 Technical Report. *arXiv* (arXiv:2303.08774). March 27, 2023. **https://arxiv.org/pdf/2303.08774.pdf**

[Ouyang et al. 2023]
Ouyang, S.; Zhang, J.M.; Harman, M.; & Wang, M. LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation. *arXiv.* August 5, 2023. **https://arxiv.org/abs/2308.02828**

[Ozkaya 2023]
Ozkaya, I. Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications. *IEEE Software.* Volume 40. Number 3. May-June 2023. Pages 4–8. **https://ieeexplore.ieee.org/document/10109345**

[Pan et al. 2023]
Pan R.; Ibrahimzada, A.R.; Krishna, R.; Sankar, D.; Wassi, L.P.; Merler, M.; Sobolev, B.; Pavuliri, R.; Sinha, S.; & Jabbarvand, R. Understanding the Effectiveness of Large Language Models in Code Translation. *arXiv.* August 6, 2023. **https://arxiv.org/pdf/2308.03109.pdf**

[Patterson et al. 2022]
Patterson, D.A.; Gonzalez, J.; Holzle, U.; Le, Q.; Liang, C.; Munguía, L.; Rothchild, D.; So, D.R.; Texier, M.; & Dean, J. The Carbon Footprint of Machine Learning Training Will Plateau, Then Shrink. *Computer.* Volume 55. Number 7. July 2022. Pages 18–28. **https://ieeexplore.ieee.org/document/9810097**

[Rastogi et al. 2023]
Charvi Rastogi, C.; Túlio Ribeiro, M.; King, N.; Nori, H.; & Amershi, S. Supporting Human-AI Collaboration in Auditing LLMs with LLMs. Pages 913–926. In *Proceedings of the 2023 AAAI/ACM Conference on AI, Ethics, and Society.* Montreal, Quebec. August 2023. **https://dl.acm.org/doi/10.1145/3600211.3604712**

[Schwartz et al. 2023]
Schwartz, S.; Yaeli, A.; & Shlo, S. Enhancing Trust in LLM-Based AI Automation Agents: New Considerations and Future Challenges. *arXiv.* August 10, 2023. **https://arxiv.org/abs/2308.05391**

[White et al. 2023]
White, Jules; Fu, Quchen; Hays, Sam; Sandborn, Michael; Olea, Carlos; Gilbert, Henry; Elnashar, Ashraf; Spencer-Smith, Jesse; and Schmidt, Douglas C. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv* (arXiv:2302.11382). February 21, 2023. **https://arxiv.org/pdf/2302.11382.pdf**

[Wolfram 2023]
Wolfram, Stephen. What Is ChatGPT Doing… and Why Does It Work? *Stephen Wolfram | Writings.* February 24, 2023. **https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/**

[Wu et al. 2023]
Wu, S.; Irsoy, O.; Lu, S.; Dabravolski, V.; Dredze, M.; Gehrmann, S.; Kambadur, P.; Rosenberg D.; & Mann, G. BloombergGPT: A Large Language Model for Finance. *arXiv.* May 9, 2023. **https://arxiv.org/pdf/2303.17564.pdf**

[Zamfirescu-Pereira et al. 2023]
Zamfirescu-Pereira, J.D.; Wong, Richmond Y.; Hartmann, Bjoern; & Yang, Qian. Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. Article 437. Pages 1–21. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (CHI '23). Hamburg, Germany. April 2023. **https://dl.acm.org/doi/abs/10.1145/3544548.3581388**

[Zhou et al. 2022]
Zhou, S.; Wang, N.; Wang, L.; Liu, H.; & Zhang, R. CancerBERT: a cancer domain-specific language model for extracting breast cancer phenotypes from electronic health records. *Journal of the American Medical Informatics Association.* Volume 29. Number 7. June 14, 2022. Pages 1208–1216. **https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9196678/**

Acknowledgements

We would like to thank Bjorn Andersson, Luiz Antunes, Nanette Brown, Anita Carleton, Douglas Schmidt, and John Robert for their suggestions and feedback.

We would like to also thank Ed Desautels and Mike Duda for editorial and design support.

About the SEI

Always focused on the future, the Software Engineering Institute (SEI) advances software as a strategic advantage for national security. We lead research and direct transition of software engineering, cybersecurity, and artificial intelligence technologies at the intersection of academia, industry, and government. We serve the nation as a federally funded research and development center (FFRDC) sponsored by the U.S. Department of Defense (DoD) and are based at Carnegie Mellon University, a global research university annually rated among the best for its programs in computer science and engineering.

Contact Us

CARNEGIE MELLON UNIVERSITY
SOFTWARE ENGINEERING INSTITUTE
4500 FIFTH AVENUE; PITTSBURGH, PA 15213-2612

sei.cmu.edu
412.268.5800 | 888.201.4479
info@sei.cmu.edu

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

DM23-2008

DOI: 10.58012/m3hj-6w28