Journal of Emerging Innovations in Engineering (JEIE)
Volume 2, Number 02, June 2026
ISSN: 3110-3375
Open Access: https://jurnal.sutajaya.org/jeie

A case study on using customized large language models for requirement-to-code transformation in software engineering

Seemal Shahbaz[1,*]

[1]Canterbury Christ Church University, Canterbury, UK
[1]Email: seemalshahbaz01@gmail.com

Article Info

Article history:
Received May 10th, 2026
Revised May 27th, 2026
Accepted June 29th, 2026

Keyword:
Prompt Engineering; Large Language Models; Software Engineering; Requirement-to-Code Transformation; AI-Assisted Code Generation.

DOI:
https://doi.org/10.65664/jeie.v2i02.22
Page: 88 - 94

Correspondent Author:
Seemal Shahbaz
Email:
seemalshahbaz01@gmail.com

ABSTRACT

Large Language Models (LLMs) have shown great promise for assisting software engineering tasks that can be automated, like mapping natural language to code. This paper describes a practical case study using a domain specific LLM to convert software requirements into semi-formal designs, tests and code. We suggest a simple iterative refinement process for engineering prompts and progressively prompting with the intention of improving the accuracy of outputs, alignment to software requirements. We validate our approach with a case study based on real-world scenarios that show how LLMs can be leveraged for requirement extraction, Generation of Object-oriented Design (OOD) and implementation steps. These findings also give an understanding of the utility of LLM-assisted development in relation to affordances and constraints.

INTRODUCTION

The recent advances in Artificial Intelligence (AI) involving Large Language Models (LLMs) have dramatically changed the domain of software engineering by allowing for smart automation of complex programming tasks. Recent advancements like GPT-based architectures have been great at natural language understanding and code generation, giving zero- or one-shot capabilities to developers enabling them to interact with systems using high-level textual input as opposed to traditional programming languages (Xie et al., 2023). This change has opened new possibilities to improve the software development lifecycle, particularly in linking software requirements to executable code. In fact, despite these advances, the difficulty of converting informal requirements into formal designs and verifiable implementations is one of the most significant problems facing software engineering and continues to be done at a very high cost by skilled humans with years of experience in a refinement fashion (Rubin et al., 2022).

Prompt engineering is a newly developed way to improve LLMs through the changing prompts instead of their architecture. Through prompt engineering, developers can shape input prompts to produce more contextually relevant and task-relevant outputs. Recent research has demonstrated that the reasoning and code generation of LLMs can be significantly improved by

ISSN: 3110-3375 DOI: https://doi.org/10.65664/jeie.v202.22

techniques like chain-of-thought prompting and few-shot learning (Pornprasit & Tantithamthavorn, 2024). In addition, research in AI-assisted software engineering helps encapsulate the increasing role of LLMs that can automate tasks like requirement analysis, design generation, and testing as a way to enhance productivity and shorten development cycles (Dilhara et al., 2024). Most previous studies concentrate on individual tasks and do not advance a unified framework that supports the complete requirement-to-code transformation process. (White et al., 2023).

This research presents the use of a custom LLM that is guided by a systematic prompt engineering process to assist with guiding an end-to-end software development pipeline. In particular, this work evaluates the usage of progressive prompting and iterative refine for transforming software requirements into functional specifications, object-oriented designs, test cases and executable codes (Li et al., 2023). Additionally, the purpose of this study is to evaluate how LLMs are applicable in real-world inspired development scenarios and which strengths and weaknesses exist when using them (Touvron et al., 2023).

This study contributes to the field of software engineering in multiple different ways. First, it proposes a structured prompt engineering framework that decomposes complex development tasks into manageable intermediate steps, improving both interpretability and output quality. Second, it shows concrete example of how a custom LLM is used in each step during the software dev life cycle (Shi et al., 2024). Third, few overviews have investigated empirical data on the working relation of software engineers and AI systems in practice as it could improve system performance (like iterative feedback). These contributions are particularly relevant for researchers and practitioners looking to integrate AI-enabled tools into mod ern development pipelines (J. Liu et al., 2022).

Our research brings together traditional isolated uses of an LLM in the software development life cycle. We combine requirements engineering, design generation, and implementation into a single prompting framework (Qiao et al., 2023). Rather than relying on the LLM to produce complete output at once, we use an interactive approach to generate results collaboratively. This interactive method enhances the quality of LLM output and gives greater transparency and control in AI-assisted tools. It also helps create a standardized, scalable LLM-centric software engineering framework for the future (Chen et al., 2021).

METHOD

The article describes an applied qualitative case study about the use of a tailored LLM to support software engineering processes, where the objective is to transform software requirements into structured artifacts such as designs and executable artefacts. The work studies a systematic process of prompt engineering followed by iterative optimization to enhance the reliability, consistency and usability of LLM-generated outputs in an environment parallelled to real world development (S. Liu et al., 2024).

Research Design

This is exploratory research, dealing with investigation of real use cases for LLMs within the software development life cycle. Domain knowledge about requirements engineering, object-oriented design and programming principles/data is fine-tuned from a general pretrained transformer. In this paper, we define the paradigm of structural prompting which partitions complex development tasks into sequenced stages to assist models with both reasoning quality and outcome quality. This is followed by an iterative process of refinement that allows for near real time improvement in outputs driven by user response (Xie et al., 2023).

Case Study Description

A Real-World Multi-Stakeholder Scheduling System Scenario with Multiple Functional Requirements and Workflow Dependencies. For this purpose, a complex and relevant case is selected to evaluate abilities of LLM in different stages/remains of Software Development Life Cycle (SDLC) including requirement analysis, system design, testing and implementation. The case represents a perfect corner to study LLM capabilities because they include structured and un-structured inputs (Pei et al., 2025).

Data Collection and Procedure

The dataset on which this study has been performed consists of multiple types of both structured semi-structured requirement documents like project scope, vision, glossary and use case

ISSN: 3110-3375 DOI: https://doi.org/10.65664/jeie.v202.22

description. We feed these inputs to the LLM, creating an authentic virtual representation of requirement engineering in real world scenarios (Zhao et al., 2021).

At each stage, the LLM produces outputs that are subsequently reviewed and iterated on through feedback. The progressive prompting methodology is helpful not only because it makes complex tasks easier to tackle step-by-step, but also because it helps keep the model's output aligned with its original goal (Sun et al., 2024).

The research procedure follows a structured prompt engineering workflow, as illustrated in Figure 1.

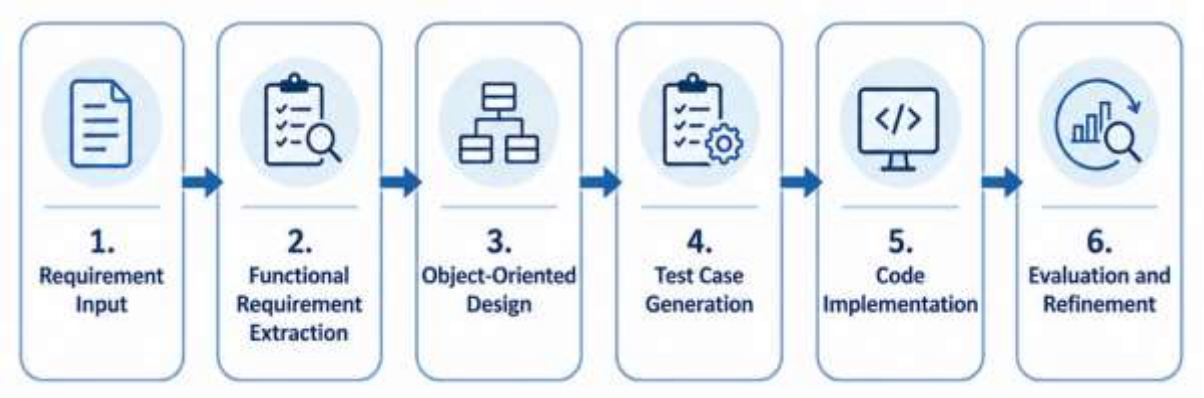

Figure 1. Research Workflow for Requirement-to-code Transformation Using LLM

Data Analysis

The method of analysing the data is qualitative, using descriptive analysis of outputs created by LLM at each development phase to determine how effectively LLM interprets software requirements and how consistently the automated design artefacts generated by LLM align with the initial requirements. We also assess the quality of code generated by LLM based on its ability to realise the design artefacts and improve implementation through iterative refinement. The results of these evaluations identify the strengths, weaknesses, and transferability of LLMs in software engineering (Han & Lyu, 2025).

To support the analysis, an evaluation framework is applied, as presented in Table 1.

Table 1. Evaluation Criteria for LLM Performance

No	Evaluation Aspect	Description
1	Requirement Accuracy	Alignment between input requirements and extracted outputs
2	Design Consistency	Logical correctness and structure of generated design
3	Code Quality	Readability, correctness, and adherence to standards
4	Iterative Improvement	Ability to refine outputs based on feedback

RESULTS AND DISCUSSION

This part provides an overview of the results of the case study and evaluates how well a custom-trained Large Language Model (LLM) converted software requirements into structured design artifacts and executable code. The outputs are the result of an iterative process between user and LLM through the prompt engineering framework utilized in methodology (Rubin et al., 2022).

Requirement Understanding and Initial Interaction

Given the LLM being 'trained' on structured requirement documents regarding the project scope, vision, glossary and usecase (just as an example), for the stage1. Figure 2: Output from the system describing its description of the uploaded documents. The first step, shows that the LLM is capable of interpreting this unstructured and semi-structured text input (in natural language) through a mapping to a coherent representation.

ISSN: 3110-3375 DOI: https://doi.org/10.65664/jeie.v202.22

On top of that, the dealing between the software engineer and the tailored LLM describes to a workflow structured as per Figure 2. The model takes the user through a series of steps to ensure that each stage of the development process is done in order. Such structured interaction minimizes ambiguity and increases clarity around the interpretation of requirements.

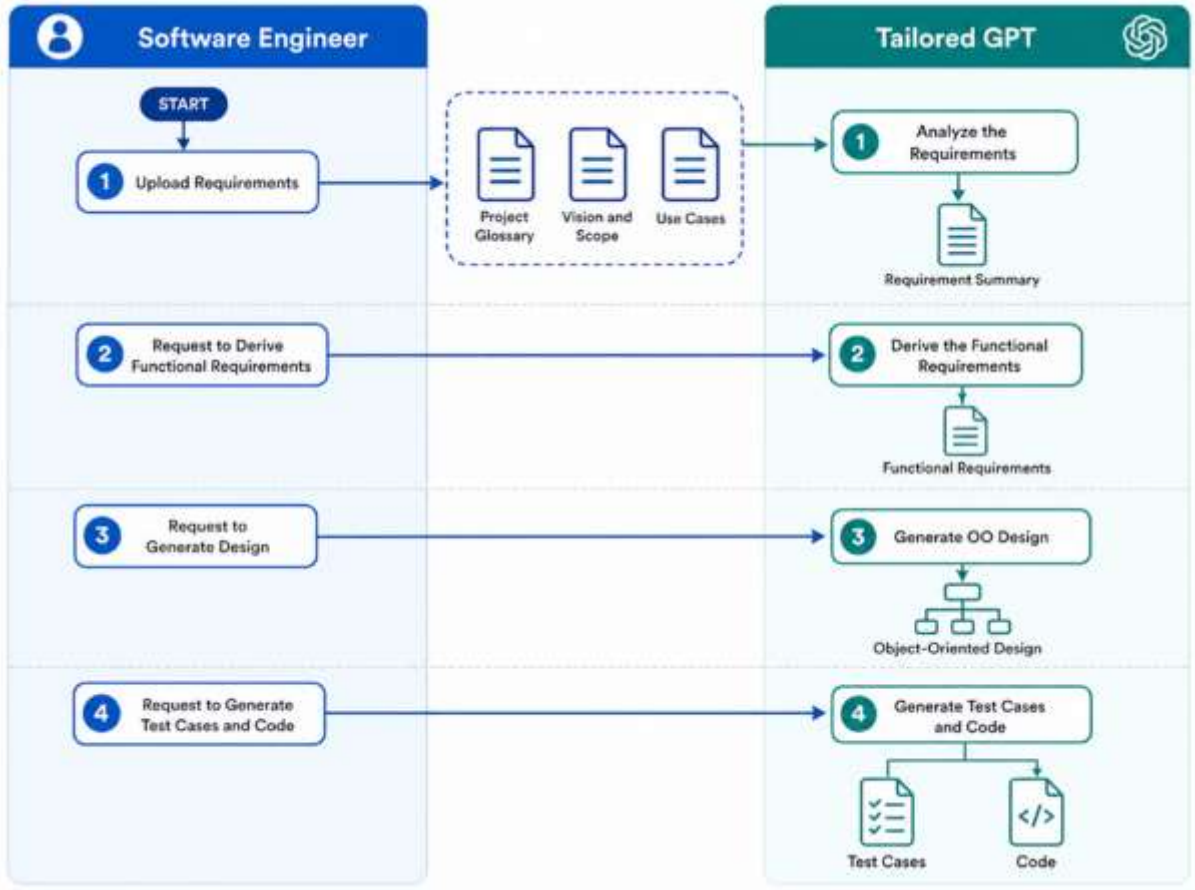

Figure 2. A Software Engineer Interaction with Tailored GPT Model

Functional Requirement Extraction

Once the input documents have been executed by this layer, an LLM creates detailed functional requirements depending on which use case was selected. The system identifies important functionalities of the system, interactions between user and system, name of expected output. This step demonstrates how the engineering of prompts allows intuitive instructions for the model to take relevant and structured requirements from a free-form description. These results suggest that the LLM is capable of converting high-level use-cases into system requirements in a suitable format. However, there were some outputs with slight ambiguities which required users to refine and specify explicitly what they wanted. This gives LLMs the ability to extract requirements sufficiently, but man needs to supervise them, they cannot fully accomplish and check the task without human verification.

Object-Oriented Design Generation

From the requirements, LLM produced an object-oriented design with class identification and attributes, methods and relationships. The model was further able to break the system down into logical parts and allocate correct duties to each class as shown by Figure 3. The produced design was reasonably good and compatible with the best practices of software engineering. Some design components needed iterative feedback and hence, refinement. This iterative method made the design much more coherent and accurate while reinforcing the need for human-in-the-loop interaction in order to achieve maximum results.

ISSN: 3110-3375 DOI: https://doi.org/10.65664/jeie.v202.22

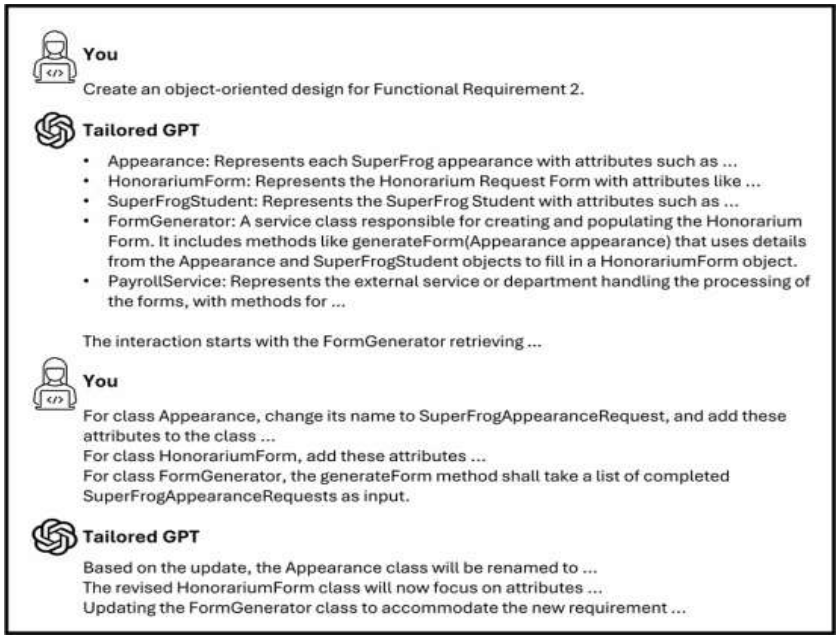

Figure 3. Object Oriented Design for One Functional Requirement

Test Case Generation and Validation

In the second step, the LLM produced test cases that were related to the system components we had designed above. These test cases were meant to confirm the functionality of the system and check whether or not it behaves as per expectations under different conditions. The model generated relevant test scenarios, although there was a need for clarification to narrow down the test cases based on actual implementation details. This step also depicts a possible application of LLMs to assist in software testing by generating validation scenarios. As with previous stages, much of the work was otherwise iteratively refining these outputs.

Code Generation and Implementation

To implement the executable code based on refined requirements and design. With the way that the LLM presented us with linear blocks of logic with very few errors handling and all at acceptable programming levels, as seen in Figure 4, the LLM's produced code was functionally equivalent to the specifications, which further provides evidence to support end-to-end development of Software using LLMs. However, the limitations found were that there are still edge cases that are not covered, and a lot of additional optimisation needs to be performed. In summary, this data shows that even though LLM can increase the speed of generating code, it is still essential to review and enhance the code through manual efforts, in order for it to meet the standards for use in production.

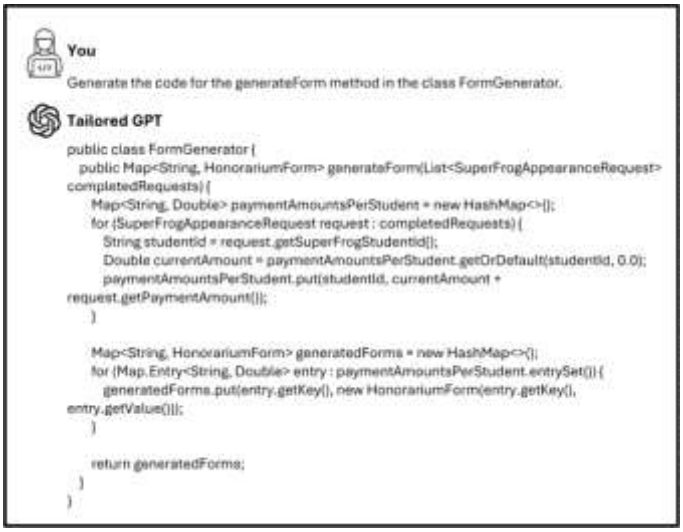

Figure 4. Implementation Code Based on Test Cases

ISSN: 3110-3375 DOI: https://doi.org/10.65664/jeie.v202.22

Workflow Efficiency and Iterative Improvement

The high-level flow of the proposed approach that describes the hierarchical setup during requirement transformation is shown in Figure 5. They learned to manipulate difficult tasks by dividing them first into simpler complete able subtasks using this step-wise prompting approach. The iterative refinement process allows for better output quality at every level. Therefore, the user had better quality based on their feedback (steering) and even prompt aspects (accuracy, consistency and completeness of results). Conclusions We believe our approach of combining structured prompting with iterative interaction is the key to achieving optimal usage of LLMs for software engineering tasks.

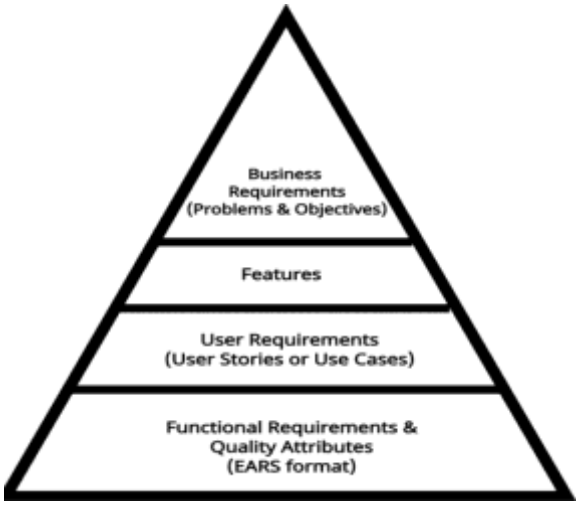

Figure 5. Workflow Hierarchy

Evaluation Summary

Lastly, TLR evaluates the quality of parts (or TLRs) of some fixed evaluation criteria of LLM output in Table 2. Experiment result show that the new change in the LLM support multiple step of software development lifecycle. And the system is extremely robust in understanding requirements, generating designs and writing code. However, the need for repetitive tuning and human supervision is still very important.

Table 2. Summary of LLM Performance Evaluation

No	Evaluation Aspect	Description
1	Requirement Accuracy	High accuracy with minor refinements required
2	Design Consistency	Moderate to high, improved through iteration
3	Code Quality	Functional but requires optimization
4	Iterative Improvement	Significant improvement observed

CONCLUSION

This paper has shipped the large language model (LLM) that assists software engineers in transforming their specifications into designs of software, test cases and real code. It draws on an external knowledge base that combines requirements engineering and software design domains. We then present a case study showcasing how this LLM can be both useful and meaningful within the context of software development. In software engineering, we also emphasized our findings that high-quality requirements are required to paper the muscles of LLM.

In the future, we will look into a wider range of techniques for specifying requirements and different methods for designing software. We will also conduct a more thorough evaluation of this new approach to further prove its effectiveness and applicability in various software engineering contexts.

REFERENCES

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. de O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., … Zaremba, W. (2021). *Evaluating Large Language Models Trained on Code*. http://arxiv.org/abs/2107.03374

Dilhara, M., Bellur, A., Bryksin, T., & Dig, D. (2024). Unprecedented Code Change Automation: The Fusion of LLMs and Transformation by Example. *Proceedings of the ACM on Software Engineering*, *1*(FSE), 631–653. https://doi.org/10.1145/3643755

Han, Y., & Lyu, C. (2025). Multi-stage guided code generation for Large Language Models. *Engineering Applications of Artificial Intelligence*, *139*, 109491. https://doi.org/10.1016/J.ENGAPPAI.2024.109491

Li, X.-Y., Xue, J.-T., Xie, Z., & Li, M. (2023). *Think Outside the Code: Brainstorming Boosts Large Language Models in Code Generation*. http://arxiv.org/abs/2305.10679

Liu, J., Shen, D., Zhang, Y., Dolan, B., Carin, L., & Chen, W. (2022). What Makes Good In-Context Examples for GPT-3? *DeeLIO 2022 - Deep Learning Inside Out: 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures, Proceedings of the Workshop*, 100–114. https://doi.org/10.18653/v1/2022.deelio-1.10

Liu, S., Wen, T., Pattamatta, A. S. L. S., & Srolovitz, D. J. (2024). A prompt-engineered large language model, deep learning workflow for materials classification. *Materials Today*, *80*, 240–249. https://doi.org/10.1016/J.MATTOD.2024.08.028

Pei, Z., Yin, J., & Zhang, J. (2025). Language models for materials discovery and sustainability: Progress, challenges, and opportunities. *Progress in Materials Science*, *154*. https://doi.org/10.1016/j.pmatsci.2025.101495

Pornprasit, C., & Tantithamthavorn, C. (2024). Fine-tuning and prompt engineering for large language models-based code review automation. *Information and Software Technology*, *175*, 107523. https://doi.org/10.1016/J.INFSOF.2024.107523

Qiao, S., Ou, Y., Zhang, N., Chen, X., Yao, Y., Deng, S., Tan, C., Huang, F., & Chen, H. (2023). Reasoning with Language Model Prompting: A Survey. *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, *1*, 5368–5393. https://doi.org/10.18653/v1/2023.acl-long.294

Rubin, O., Herzig, J., & Berant, J. (2022). Learning To Retrieve Prompts for In-Context Learning. *NAACL 2022 - 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference*, 2655–2671. https://doi.org/10.18653/v1/2022.naacl-main.191

Shi, Z., Wei, J., Xu, Z., & Liang, Y. (2024). Why Larger Language Models Do In-context Learning Differently? *Proceedings of Machine Learning Research*, *235*, 44991–45013.

Sun, Y., Li, X., Liu, C., Deng, X., Zhang, W., Wang, J., Zhang, Z., Wen, T., Song, T., & Ju, D. (2024). Development of an intelligent design and simulation aid system for heat treatment processes based on LLM. *Materials and Design*, *248*. https://doi.org/10.1016/j.matdes.2024.113506

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. (2023). *LLaMA: Open and Efficient Foundation Language Models*. http://arxiv.org/abs/2302.13971

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., & Schmidt, D. C. (2023). *A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT*. http://arxiv.org/abs/2302.11382

Xie, T., Wan, Y., Huang, W., Zhou, Y., Liu, Y., Linghu, Q., Wang, S., Kit, C., Grazian, C., Zhang, W., & Hoex, B. (2023). *Large Language Models as Master Key: Unlocking the Secrets of Materials Science with GPT*. http://arxiv.org/abs/2304.02213

Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K. J., Ajagbe, M. A., Chioasca, E.-V., & Batista-Navarro, R. T. (2021). Natural Language Processing for Requirements Engineering: A Systematic Mapping Study. *ACM Comput. Surv.*, *54*(3). https://doi.org/10.1145/3444689