

Open-Source LLMs for Technical Q&A: Lessons from StackExchange

Zeerak Babar, Nafiz Imtiaz Khan, Muhammad Hassnain, and Vladimir Filkov

University of California at Davis, Davis, CA, USA
{zebabar, nikhan, mhassnain, vfilkov}@ucdavis.edu

Abstract. In the rapidly evolving domain of software engineering (SE), Large Language Models (LLMs) are increasingly leveraged to automate developer support. Open source LLMs have grown competitive with proprietary models such as GPT-4 and Claude-3, without the associated financial and accessibility constraints. This study investigates whether state of the art open source LLMs including Solar-10.7B, CodeLlama-7B, Mistral-7B, Qwen2-7B, StarCoder2-7B, and LLaMA3-8B can generate responses to technical queries that align with those crafted by human experts. Leveraging retrieval augmented generation (RAG) and targeted fine tuning, we evaluate these models across critical performance dimensions, such as semantic alignment and contextual fluency. Our results show that Solar-10.7B, particularly when paired with RAG and fine tuning, most closely replicates expert level responses, offering a scalable and cost effective alternative to commercial models. This vision paper highlights the potential of open-source LLMs to enable robust and accessible AI-powered developer assistance in software engineering.

Keywords: Large Language Models (LLMs) · Open Source Models · Retrieval-Augmented Generation (RAG) · Technical Queries · StackExchange

1 Introduction

Large Language Models (LLMs) are built on complex architectures and trained on extensive datasets, enabling them to generate human-like text that can answer questions, write essays, summarize content, and engage in conversations [49]. Furthermore, by using tailored prompts, the output can be adjusted to mimic the behavior of specific demographic profiles. As a result, the advent of LLMs offers a valuable opportunity to simulate software developers, addressing traditional challenges associated with recruiting human developers for coding assistance/research purposes.

In the current environment of ever-increasing expectations, software developers are starting to rely on LLMs to obtain immediate and insightful answers to software development-related questions and solve technical challenges [10, 21]. For instance, the study by Oishwee et al. [32] analyzed whether ChatGPT can answer Android permission-related questions. The authors analyzed 1,008 StackOverflow

questions and their accepted answers, finding that 50% of ChatGPT-generated responses aligned with those accepted by developers. In another study, Gerosa et al. [16] proposed leveraging LLMs to generate synthetic qualitative data, suggesting that AI could potentially replace human participants in software engineering research. However, they caution that ethical concerns, model biases, and authenticity issues remain significant challenges. Separately, Haque [17] argues that LLMs are not only reshaping software development but also redefining the developer’s role. While challenges persist, LLMs present unprecedented opportunities for innovation and collaboration, making early adoption essential for remaining competitive in this rapidly evolving field.

While LLMs are widely used by developers and researchers, most current research focuses on proprietary LLMs like `ChatGPT`, `Gemini` and `Claude-3` [44, 28]. These models, although powerful, come with high financial costs, often making them inaccessible to many developers and researchers. In this scenario, Open-source LLMs, which can be run locally and are free to use, offer a promising alternative for broader accessibility.

A Retrieval-Augmented Generation (RAG) [15] framework could further enhance LLM-generated responses by supplying relevant contextual information alongside user queries. However, no previous studies have explored how the RAG framework can be implemented, what type of context should be provided for individual user queries, or the effectiveness of the RAG framework in this specific task. Additionally, fine-tuning LLMs [11] may improve their ability to answer developer questions accurately, yet no studies have explored the combined effects of fine-tuning and the RAG framework in this context.

Moreover, although some research has examined question-answering platforms like `StackOverflow` [12] and `Reddit` [36], there has been no focused analysis of the broader community network `StackExchange` [12], which consists of 183 Q&A sites [12], including `StackOverflow`, `Super User`, and `Game Development`. The platform includes 60 technical sites out of a total of 183, such as `Data Science`, `Web Development`, and `Information Security`. The vast majority of contributors to these technical sites are developers, sharing their expertise and solving coding-related problems [40].

Focusing on the research gaps, we have developed three research questions:

- RQ₁:** Can local LLMs Effectively Answer Technical Questions in a Base Setting?
RQ₂: To what extent does the RAG framework enhance LLM response quality by incorporating additional contextual information?
RQ₃: Can Fine-Tuning Within the RAG Framework Further Improve Model Performance?

The study is structured to address the research questions outlined above, specifically examining whether free, open source models can provide developers with equally valuable insights.

2 Study Methodology

2.1 Data Collection

For collecting data, we identified StackExchange [4] data source containing developer-posed questions alongside human responses. Rather than directly scraping data from this platform, we leveraged the publicly available StackExchange Data Dump [5], a downloadable archive that provides snapshots of the platform’s content at specific intervals. This dataset includes various elements such as questions, answers, comments, user profiles, and tags, structured in formats like XML. Spanning from May 2014 to December 2023, the gathered dataset comprises 73,560 posts and 81,283 comments contributed by 54,037 distinct authors. Out of 73,560 posts, we randomly collected 41,000 posts and divided them into three subsets: 1) *Testing Dataset*: 1,000 samples used exclusively for testing; 2) *RAG Database*: 20,000 samples for the RAG framework; and 3) *Training Dataset*: 20,000 samples for fine-tuning the selected LLM model.

2.2 Data Synthesis

The **StackExchange** dataset underwent pre-processing to ensure data quality. First, each post included multiple comments as potential solutions to developer queries. Using the **AcceptedAnswer** tag, we identified the most relevant human response, selecting only posts with an accepted answer to create a reliable question-answer dataset for comparison with LLM responses. Next, we removed unnecessary HTML tags from posts and comments to maintain a clean text format for analysis.

2.3 Selecting the LLM Models

In this study, we required models capable of generating both natural language and code, suitable for question-answering tasks grounded in technical software engineering contexts. Based on prior work in code-related question-answering [38, 23, 27], we selected a set of high-performing, open-source language models with approximately 7-10 billion parameters. Our selection prioritized models that support general-purpose instruction following, code generation, and contextual reasoning—making them viable candidates for integration into RAG pipelines. The selected models are briefly described below:

Mistral-7B: **Mistral-7B** is a 7-billion-parameter dense transformer model developed by Mistral-AI [20, 2]. Known for its competitive performance across reasoning and generation tasks, it remains efficient and accurate despite its relatively compact size. We use the "Mistral-7B Instruct" variant, which is fine-tuned for instruction-following and conversational use cases [1].

Solar-10.7B: **Solar-10.7B** is a 10.7-billion-parameter model that balances performance across both code and text generation tasks. Prior studies have shown its strength in semantic reasoning and synthesis, making it a top choice for coding-related QA [23].

CodeLlama-7B: CodeLlama-7B is an open-source model specialized for code-related tasks. Developed by Meta, it provides strong support for multi-language programming and accurate code synthesis [38]. While optimized for code generation, it is also capable of producing well-structured natural language responses.

Qwen2-7B: Qwen2-7B, developed by Alibaba, is a general-purpose transformer with strong multilingual and reasoning capabilities [29]. It performs competitively across standard NLP benchmarks and is particularly well-suited for retrieval-augmented tasks due to its robustness in context-grounded generation.

StarCoder2-7B: StarCoder2-7B, jointly developed by Hugging Face and ServiceNow, is trained on permissively licensed code and instruction datasets. While it excels at code generation, it also supports high-quality natural language output, making it suitable for technical QA workflows involving both modalities [30].

LLaMA-3 8B: LLaMA-3 8B delivers state-of-the-art performance for its size. Its instruction-tuned variant demonstrates notable improvements in coherence and factual accuracy [47].

These chosen models represent the most recent and widely adopted 7-10B range open-source models with strong instruction-following capabilities. Fine-tuning was selectively applied to the best-performing model due to resource constraints and its superior baseline performance.

2.4 Prompt Design

A well-defined prompt is essential for an LLM to generate high-quality responses [46]. The prompt guides the LLM to align the query with context, ensuring clarity, relevance, and integration of technical details. In this study, we used a vanilla prompting technique for all experiments, outlined as follows:

Listing 1.1. Prompt Template

```

1 You are a helpful assistant with expertise in coding and
  technical topics. Use the following context to answer
  the question as accurately as possible, especially
  focusing on technical details and code examples if
  relevant.
2 Context:<<CONTEXT>>
3 Question:<<QUESTION>>
4 Provide a helpful and accurate answer, focusing on coding
  and technical topics.

```

2.5 Experimental Design

Base Configuration The base setting involves providing only the query to the model and eliciting responses from models. To obtain responses, we have used a tool named `Ollama` [3], which enables the local execution of open-source models. Each of the questions of the dataset is passed to the LLM models as a prompt, and the obtained responses are recorded in local machine.

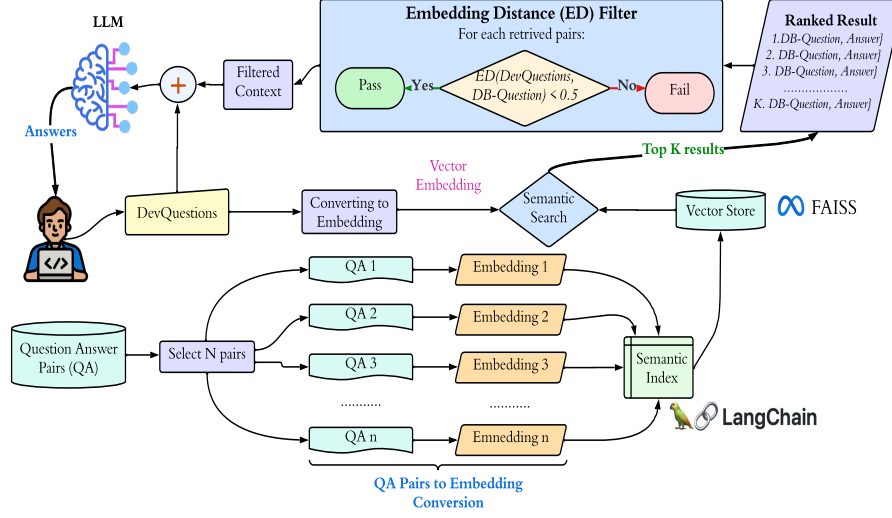


Fig. 1. Architecture of the proposed RAG pipeline

RAG Configuration We set up an RAG pipeline to supply additional context to the LLM with each question. The proposed RAG architecture, shown in Figure 1, includes interconnected components that efficiently retrieve and utilize contextual information from a vector database.

Initially, each of the question-answer pairs is converted into documents. The purpose of this transformation is to prepare the data for vectorization. Next, each of the documents is converted into embeddings. For embedding generation, **all-MiniLM-L6-v2** sentence transformer model has been used [39], which is based on the **MiniLM** architecture and is designed to balance performance and speed [42]. The design considerations make the model suitable for tasks such as semantic search, clustering, and classification [42].

Next, the pipeline leverages **LangChain** [25] to build semantic indexes for each embedding, facilitating efficient retrieval and ranking of information based on meaning and context, rather than relying solely on keyword matching [33]. These embeddings, along with their semantic indexes, are stored in **FAISS** [31], which is a high-performance vector database tailored for similarity search.

When a user submits a query, it is converted into an embedding using the sentence transformer model (**all-MiniLM-L6-v2**). The pipeline performs a semantic search within the vector database using this embedded query, retrieving the most relevant indexes ($n=5$). The plain text of these indexes is then appended to the user’s query and sent to the model. Since some retrieved contexts may be irrelevant, we evaluate the similarity between the query ($\mathbf{q}$) and the retrieved indexes ($\mathbf{d}$) using an embedding distance evaluator [24]. This evaluator calculates a relevance score to ensure only relevant question-answer pairs are included. If $S(\mathbf{q}, \mathbf{d}) > 0.5$, the context is excluded, as a higher similarity score suggests

misalignment with the original query and the retrieved content from the vector database.

The similarity score S is defined as:

$$S(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q} \cdot \mathbf{d}}{\|\mathbf{q}\| \|\mathbf{d}\|}$$

Fine-Tuned Model with RAG Configuration In this configuration, we fine-tuned the model (the best-performing LLM from the *Base* and *RAG* configurations) and used it within the RAG architecture. The fine-tuned model improves the generative component, while the RAG framework ensures contextually relevant responses by retrieving information from a domain-specific knowledge base. The model fine-tuning process is outlined in sub-section 2.6.

Experimental Consideration In our study, all experiments were conducted with a constant model temperature ($temp = 0$). Temperature is a hyper-parameter that controls the randomness of an LLM’s output [37]. A lower temperature makes responses more deterministic and focused, while a higher temperature increases randomness and creativity. Other hyper-parameters, such as top_p , top_k , min_p , and $repeat_penalty$, were kept at their default values.

2.6 Fine-Tuning LLM

The fine-tuning process, along with the selected hyperparameters, is illustrated in Figure 2. We employed the Low-Rank Adaptation (LoRA) [19] technique to fine-tune the best-performing model on 20,000 question-answer pairs, using the test set ($n=1,000$) as the validation set. The fine-tuning parameters included a batch size of 8, AdamW [13] as the optimizer, a learning rate of $2.5e-5$, and 500 training steps. The `q_proj` layer was specifically fine-tuned, and early stopping [34] was applied as the stopping criterion to ensure optimal performance. Furthermore, we utilized 4-bit quantization combined with a mixed-precision configuration using `bfloat16`. This setup significantly minimized the model’s memory footprint without compromising its performance.

2.7 Evaluating the responses of LLMs

We selected three quantitative evaluation metrics based on prior literature [22]. These metrics allow assessment of the LLM-generated responses using gold annotations as references. The selected metrics include one text-similarity-based metric (*Cosine Similarity*) and two semantic-based metrics (*BERTScore* and *METEOR*). Semantic-based metrics analyze meaning and context, emphasizing coherence and relevance [18], whereas similarity-based metrics focus primarily on lexical overlap or statistical similarity between texts [14]. Although no fixed threshold universally defines good performance, recent studies from top-ranked

venues suggest that a *BERTScore* above 0.5, a *Cosine Similarity* score above 0.3, and a *METEOR* score above 0.5 are indicative of strong performance [6–8, 41].

Cosine Similarity. It calculates the cosine of the angle between two vectors to assess similarity, quantifying the semantic closeness between human and LLM-generated responses [35].

BERTScore. It uses embeddings from pre-trained BERT models to measure semantic similarity between machine-generated and reference text, comparing cosine similarity of embeddings to quantify alignment in meaning and context with human-written references [48].

METEOR Score. It stands for Metric for Evaluation of Translation with Explicit Ordering, which assesses both lexical and semantic similarity, incorporating synonymy, stemming, and word order for a nuanced alignment between generated content and human references [9].

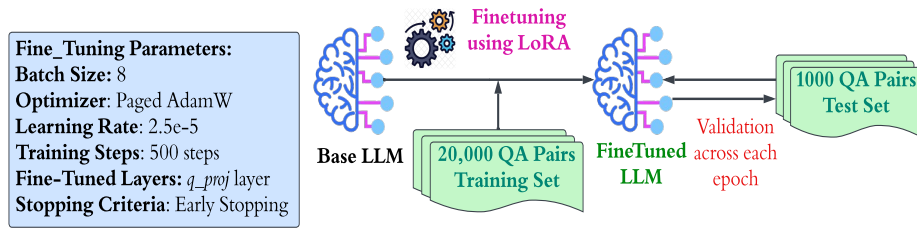


Fig. 2. Overview of the fine-tuning process, along with the selected hyper-parameters

3 Results & Discussion

The results of the experiments of this study are shown in Table 1. This section answers the questions of the stated RQs and discusses the implications of the results.

RQ1: Can local LLMs Effectively Answer Technical Questions in a Base Setting?

To address RQ1, base-setting experiments were conducted across a set of local models. Among all evaluated models, *Solar-10.7B* consistently outperformed others, achieving the highest **Cosine Similarity** (0.56) and **BERTScore** (0.54) in the base configuration. Its superior alignment with human-authored answers highlights its capability in understanding and replicating technical semantics effectively.

Other models such as *LLaMA-3 8B* (0.55 cosine, 0.53 BERT) and *CodeLlama-7B* (0.54 cosine, 0.54 BERT) followed closely behind, indicating strong but slightly less reliable performance. *Qwen2-7B* and *StarCoder2-7B* also showed competitive results.

Mistral-7B, on the other hand, underperformed in the base setting, especially with a **Cosine Similarity** of 0.32 and a notably lower **BERTScore** of 0.41. This

Models	Base Setting			RAG Architecture			RAG+Fine-Tuned Model		
	Cosine-Similarity	BERT	METEOR	Cosine-Similarity	BERT	METEOR	Cosine-Similarity	BERT	METEOR
CodeLlama-7B	0.54	0.54	0.22	0.54	0.55	0.25	X	X	X
Solar-10.7B	0.56	0.54	0.23	0.64	0.59	0.31	0.65	0.63	0.61
Mistral-7B	0.32	0.41	0.11	0.57	0.55	0.26	X	X	X
StarCoder2-7B	0.50	0.52	0.20	0.58	0.56	0.28	X	X	X
Qwen2-7B	0.53	0.53	0.22	0.60	0.57	0.29	X	X	X
LLaMA 3 8B	0.55	0.53	0.23	0.61	0.58	0.30	X	X	X

Table 1. Comprehensive overview of the performance of local language models across three configurations: Base Setting, RAG Configuration, and RAG and Fine-Tuned Configuration. In the RAG+Fine-Tuned configuration, "X" indicates that a particular model was not fine-tuned.

suggests limited effectiveness in capturing the semantic intent of the technical responses.

Across all models, METEOR scores remained relatively low in the base setting (ranging from 0.11 to 0.23), pointing to ongoing challenges in achieving fluent and well-structured natural language generation without context-aware enhancements.

RQ₁ Findings: *Solar-10.7B* remains the top-performing model in the base setting, closely followed by *LLaMA-3 8B* and *CodeLlama-7B*. While newer 7B models like *Qwen2-7B* and *StarCoder2-7B* show promising capabilities, low METEOR scores across the board highlight limitations in linguistic fluency that could benefit from contextual or architectural augmentation.

RQ2: To what extent does the RAG framework enhance LLM response quality by incorporating additional contextual information?

To explore RQ2, we evaluated the effectiveness of integrating a RAG framework in enhancing response quality. As shown in Table 1, all evaluated models demonstrated performance improvements in the **RAG Architecture** configuration, benefiting from access to relevant contextual information.

Solar-10.7B exhibited the most substantial gains, with its **Cosine Similarity** increasing from 0.56 to 0.64 and **BERTScore** rising to 0.59. These results indicate stronger semantic alignment and better preservation of the intended meaning in technical responses. The METEOR score also improved notably, reaching 0.31, suggesting enhanced fluency and coherence under the RAG framework.

Other models, including *LLaMA-3 8B* (0.61 cosine, 0.58 BERT), *Qwen2-7B* (0.60 cosine, 0.57 BERT), and *StarCoder2-7B* (0.58 cosine, 0.56 BERT), also showed meaningful improvements compared to their base performance. *Mistral-7B*, despite having the weakest base scores, experienced a sharp increase in **Cosine Similarity** from 0.32 to 0.57 under RAG, highlighting RAG’s corrective effect on underperforming models.

RQ₂ Findings: The RAG framework significantly improves performance across all models by enriching responses with relevant contextual content. *Solar-10.7B* benefits the most, while even weaker models like *Mistral-7B* see large relative gains, underscoring RAG’s potential to enhance both semantic fidelity and linguistic fluency.

RQ3: Can Fine-Tuning Within the RAG Framework Further Improve Model Performance?

We fine-tuned the *Solar-10.7B* model, which we found as the best performed LLM from base and RAG configurations. Alike other settings, fine-tuning under the RAG framework showed substantial improvements. Fine-tuning resulted in increased **BERTScore** and **METEOR**, enhancing the model’s ability to generate responses with both semantic richness and fluency. While **Cosine Similarity** showed minor changes from the RAG-only setting, the gains in **BERTScore** and **METEOR** indicate that fine-tuning refines the model’s response quality.

RQ₃ Findings: The RAG framework with a fine-tuned model yields additional performance gains, indicating that their integration enhances the quality and relevance of the generated insights, leading to more accurate and contextually appropriate results.

4 Threats to Validity

Our study has several limitations that highlight directions for future work. First, the dataset size for different experiments was relatively small. Future studies could improve reliability by using larger datasets. Second, we analyzed only six LLM models within a limited parameter range; expanding to a wider variety, especially larger models, could offer deeper insights into model performance. Third, we used a basic prompting technique, without advanced in-context approaches like **Chain-of-Thought** [43], **Tree-of-Thought** [45], or **n-shot prompting** [26]. Future research exploring these techniques may enhance model reasoning and response quality.

5 Conclusion

This study examined the viability of using locally deployed, fine-tuned LLMs as alternatives to proprietary models for SE tasks. By addressing three research questions, we assessed six open source models in different configurations: a base setting, a RAG framework, and fine-tuning within RAG. Our results show that *Solar-10.7B*, a locally deployed model, aligned well with human responses, especially with RAG and fine-tuning, excelling in technical inquiries on Stack-Exchange. RAG integration enhanced performance, while fine-tuning further improved contextual fluency. This vision paper demonstrates that open-source LLMs, optimized with RAG and fine-tuning, present a promising, cost-effective alternative for SE tasks.

Acknowledgement

This research was supported by the National Science Foundation under Grant No. 2020751, as well as by the Alfred P. Sloan Foundation through the OSPO for UC initiative (Award No. 2024-22424)

References

1. Mistral-7b instruct v0.2 on hugging face, <https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2>, accessed: 2024-11-08
2. Mistral ai, <https://mistral.ai/>, accessed: 2024-11-08
3. Ollama, <https://ollama.com/>, accessed: 2024-11-08
4. Stack exchange - where developers learn, share, build careers, <https://stackoverflow.com/>, accessed: 2024-11-08
5. Stack exchange archive on the internet archive, <https://archive.org/details/stackoverflow>, accessed: 2024-11-08
6. Ahmed, T., Bird, C., Devanbu, P., Chakraborty, S.: Studying llm performance on closed-and open-source data. arXiv preprint arXiv:2402.15100 (2024)
7. Ahmed, T., Pai, K.S., Devanbu, P., Barr, E.: Automatic semantic augmentation of language model prompts (for code summarization). In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. pp. 1–13 (2024)
8. Al-Kaswan, A., Ahmed, T., Izadi, M., Sawant, A.A., Devanbu, P., van Deursen, A.: Extending source code pre-trained language models to summarise decompiled binaries. In: 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). pp. 260–271. IEEE (2023)
9. Banerjee, S., Lavie, A.: Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In: Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization. pp. 65–72 (2005)
10. Chen, X., Gao, C., Chen, C., Zhang, G., Liu, Y.: An empirical study on challenges for llm developers. arXiv preprint arXiv:2408.05002 (2024)
11. Dodge, J., Ilharco, G., Schwartz, R., Farhadi, A., Hajishirzi, H., Smith, N.: Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. arXiv preprint arXiv:2002.06305 (2020)
12. Exchange, S.: Stack exchange. <https://stackoverflow.com/>, accessed: 2024-11-09
13. Face, H.: Adamw optimizer (2025), <https://huggingface.co/docs/bitsandbytes/en/reference/optim/adamw>, accessed: 2025-01-17
14. Finnie, G., Sun, Z.: Similarity and metrics in case-based reasoning. International journal of intelligent systems **17**(3), 273–287 (2002)
15. Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., Wang, H.: Retrieval-augmented generation for large language models: A survey. arXiv preprint arXiv:2312.10997 (2023)
16. Gerosa, M., Trinkenreich, B., Steinmacher, I., Sarma, A.: Can ai serve as a substitute for human subjects in software engineering research? Automated Software Engineering **31**(1), 13 (2024)
17. Haque, M.A.: Llms: A game-changer for software engineers? arXiv preprint arXiv:2411.00932 (2024)

18. Hu, B., Kalfoglou, Y., Alani, H., Dupplaw, D., Lewis, P., Shadbolt, N.: Semantic metrics. In: *Managing Knowledge in a World of Networks: 15th International Conference, EKAW 2006, Poděbrady, Czech Republic, October 2-6, 2006. Proceedings* 15. pp. 166–181. Springer (2006)
19. Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W.: Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021)
20. Jiang, A.Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D.S., Casas, D.d.l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al.: Mistral 7b. *arXiv preprint arXiv:2310.06825* (2023)
21. Jin, H., Huang, L., Cai, H., Yan, J., Li, B., Chen, H.: From llms to llm-based agents for software engineering: A survey of current, challenges and future. *arXiv preprint arXiv:2408.02479* (2024)
22. Khan, N.I., Filkov, V.: Evidencebot: A privacy-preserving, customizable rag-based tool for enhancing large language model interactions. *FSE Companion '25: Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering* (2025)
23. Kim, D., Park, C., Kim, S., Lee, W., Song, W., Kim, Y., Kim, H., Kim, Y., Lee, H., Kim, J., et al.: Solar 10.7 b: Scaling large language models with simple yet effective depth up-scaling. *arXiv preprint arXiv:2312.15166* (2023)
24. LangChain: Embedding distance evaluation (2024), https://python.langchain.com/v0.1/docs/guides/productionization/evaluation/string/embedding_distance/
25. LangChain: Langchain - building applications with llms. <https://www.langchain.com/> (2024), accessed: 2024-09-06
26. Lee, U., Jung, H., Jeon, Y., Sohn, Y., Hwang, W., Moon, J., Kim, H.: Few-shot is enough: exploring chatgpt prompt engineering method for automatic question generation in english education. *Education and Information Technologies* **29**(9), 11483–11515 (2024)
27. Li, L., He, X., Wang, H., Wang, L., He, L.: How do humans write code? large models do it the same way too. *arXiv preprint arXiv:2402.15729* (2024)
28. Liang, J.T., Badea, C., Bird, C., DeLine, R., Ford, D., Forsgren, N., Zimmermann, T.: Can gpt-4 replicate empirical software engineering research? *Proceedings of the ACM on Software Engineering* **1**(FSE), 1330–1353 (2024)
29. Liu, W., Bu, T., Yu, E., Li, D., Ai, D., Lu, Z., Luo, H.: Optimizing few-shot learning: From static to adaptive in qwen2-7b. In: *Amazon KDD Cup 2024 Workshop* (2024)
30. Lozhkov, A., Li, R., Allal, L.B., Cassano, F., Lamy-Poirier, J., Tazi, N., Tang, A., Pykhtar, D., Liu, J., Wei, Y., et al.: Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173* (2024)
31. Meta AI: Faiss: Facebook ai similarity search, <https://ai.meta.com/tools/faiss/>, accessed: 2024-11-08
32. Oishwee, S.J., Stakhanova, N., Codabux, Z.: Large language model vs. stack overflow in addressing android permission related challenges. In: *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*. pp. 373–383. IEEE (2024)
33. Papadimitriou, C.H., Tamaki, H., Raghavan, P., Vempala, S.: Latent semantic indexing: A probabilistic analysis. In: *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. pp. 159–168 (1998)
34. Prechelt, L.: Early stopping-but when? In: *Neural Networks: Tricks of the trade*, pp. 55–69. Springer (2002)

35. Rahutomo, F., Kitasuka, T., Aritsugi, M., et al.: Semantic cosine similarity. In: The 7th international student conference on advanced science and technology ICAST. vol. 4, p. 1. University of Seoul South Korea (2012)
36. Reddit: Reddit. <https://www.reddit.com/?rdt=40286>, accessed: 2024-11-09
37. Renze, M., Guven, E.: The effect of sampling temperature on problem solving in large language models. arXiv preprint arXiv:2402.05201 (2024)
38. Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X.E., Adi, Y., Liu, J., Sauvestre, R., Remez, T., et al.: Code llama: Open foundation models for code. arXiv preprint arXiv:2308.12950 (2023)
39. Sentence-Transformers: all-minilm-l6-v2 on hugging face, <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>, accessed: 2024-11-08
40. Tahir, A., Dietrich, J., Counsell, S., Licorish, S., Yamashita, A.: A large scale study on how developers discuss code smells and anti-pattern in stack exchange sites. *Information and Software Technology* **125**, 106333 (2020)
41. Virk, Y., Devanbu, P., Ahmed, T.: Enhancing trust in llm-generated code summaries with calibrated confidence scores. arXiv preprint arXiv:2404.19318 (2024)
42. Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., Zhou, M.: Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems* **33**, 5776–5788 (2020)
43. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V., Zhou, D., et al.: Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* **35**, 24824–24837 (2022)
44. Xiao, T., Treude, C., Hata, H., Matsumoto, K.: Devgpt: Studying developer-chatgpt conversations. In: 2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR). pp. 227–230. IEEE (2024)
45. Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., Narasimhan, K.: Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems* **36** (2024)
46. Zamfirescu-Pereira, J., Wong, R.Y., Hartmann, B., Yang, Q.: 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 (2023)
47. Zhang, D., Huang, X., Zhou, D., Li, Y., Ouyang, W.: Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b. arXiv preprint arXiv:2406.07394 (2024)
48. Zhang, T., Kishore, V., Wu, F., Weinberger, K.Q., Artzi, Y.: Bertscore: Evaluating text generation with bert. arXiv preprint arXiv:1904.09675 (2019)
49. Zhao, W.X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al.: A survey of large language models. arXiv preprint arXiv:2303.18223 (2023)