FlowState: Composing foundation models and retrieval for issue priority level prediction

Stanford CS224N Custom Project

Alex Gilbert
Department of Electrical Engineering
Stanford University
alexrg@stanford.edu

Gustavs Zilgalvis
Freeman Spogli Institute for International Studies
Stanford University
gustavs@stanford.edu

Abstract

Khattab et al. (2023) demonstrate 37–120% relative gains on the SQuAD, HotPotQA, and QReCC evaluations with a language model (LM) with retrieval compared with a vanilla LM. In this paper, we test the hypothesis that implementing fine-tuning and retrieval for foundation models (FMs) yields significant improvements on the task of predicting the priority of issues within issue tracking systems (ITS) compared with vanilla FMs. To test this hypothesis, we curate a novel dataset and develop an evaluation for this task called *FlowTest*. We found that with retrieval, GPT-3.5 Turbo, GPT-4 Turbo, and GPT-4o achieved relative gains of 1.2%, 3.0%, and 7.4% over the vanilla FMs. The fine-tuned GPT-3.5 Turbo model and the fine-tuned model with retrieval achieved relative gains of 3.20% and 3.32% over the accuracy of GPT-3.5 Turbo. These findings suggest that retrieval provides a modest boost, particularly for larger models with strong generalization capabilities, while fine-tuning offers significant improvements, likely due to enhanced local knowledge about the task. As models scale, we expect retrieval to become more important, but for fine-tuning to remain critical. We believe that our approach could offer significant utility to organizations with ITS processes.

1 Key Information

- Mentor: Rashon Poole

2 Introduction

For software teams around the world, issue tracking systems (ITSs) have become indispensable tools for streamlining workflow efficiency. However, there is a significant degree of subjectivity involved in issue tracking, requiring regular team discussions and substantial effort devoted to aligning perceptions and ensuring consistency. In this work, we focus on the assignment of issue priorities—a primary source of ambiguity and overhead in ITSs—and investigate the effectiveness of foundation models (FMs) as a tool to alleviate this burden and mitigate the effects of cross-team discrepancies. Specifically, we consider the task of classifying task priority level in alignment with organizational standards, using its textual description and metadata.

Khattab et al. (2023) demonstrate 37–120% relative gains on the Stanford Question Answering Dataset (SQuAD), HotPotQA, and Question Rewriting in Conversational Context (QReCC) evaluations with a language model with retrieval and optimized in-context learning, compared with a vanilla language model (GPT-3.5). Motivated by these findings, we hypothesize that implementing fine-tuning and retrieval using a relevant ITS database should yield significant improvements on our target task of priority prediction, relative to vanilla FM baselines.

To test our hypothesis, we first curate a novel dataset and create an evaluation (*FlowTest*) for ITS priority prediction, based on the public ITS repositories presented in Montgomery et al. (2022). We

proceed to implement two key methods for improved priority prediction: an ITS-based retrieval-augmented generation (RAG) pipeline, and priority prediction fine-tuning. We proceed to run an extensive set of experiments: first evaluating our *FlowTest* evaluation on three state-of-the-art vanilla FMs (GPT-3.5 Turbo, GPT-4 Turbo, and GPT-4o), before testing out the effectiveness of retrieval and fine-tuning for improving our baseline benchmark results. We believe that our approach could offer significant utility to real-world organizations looking to streamline their ITS processes.

3 Related Work

For decades, issue tracking services have attracted significant attention from software researchers looking to improve the procedures used by engineering teams. Automation of various aspects of issue tracking has been a particularly common area of investigation, for example predicting the ideal issue assignee Jeong et al. (2009), identifying duplicate issues Deshmukh et al. (2017); He et al. (2020); Wang et al. (2008), or establishing issue linkages Lüders et al. (2022). Most closely related to our task, the researchers in Lamkanfi et al. (2010, 2011) present several techniques for predicting *bug severity* levels from language descriptions using classical machine learning and NLP. While they see some success in this domain, severity is less subjective and less critical to planning than priority, and their use of classical methods limits generalization and necessitates additional engineering. Moreover, bug reports are a relatively narrow class of issues which only apply to a small subset of software teams. The authors of Montgomery et al. (2022) had this last point in mind when they introduced their extremely large-scale accumulation of public Jira data for more universally applicable ITS research. Here, we utilize this newly accessible data along with the dramatic recent NLP advances to create a system with genuine value for a wide range of software and other projects.

The past few years have seen unprecedented progress in the field of NLP, and powerful, large-scale foundation models (FMs), such as Claude (a FM released by Anthropic in March 2024), Gemini (DeepMind (2024)), and GPT (3, 4, and so on) (OpenAI (2020, 2024)), have become ubiquitous. While their value as general tools for a diverse range of tasks is impressive, numerous techniques have been developed for designing systems which can enhance the performance of FMs for a particular task. The advent of retrieval augmented generation (Khattab et al. (2021)) is a key improvement on vanilla FMs, enabling the grounding of outputs in documents from a relevant database, thereby helping to avoid hallucinations on domain specific data. Fine-tuning on task-specific data is another technique which, when applied to FMs, can greatly benefit their performance on that test. In Khattab et al. (2023), the authors perform a detailed comparison between multi-stage systems of retrieval and prediction with varying degrees of complexity. They find that even a simple, single step *retrieve-then-read* pipeline (where the retrieved passage is simply appended to the prompt context) yields a nearly 75% improvement on *open domain question answering* (with even greater improvements from multi-hop systems with optimized interactions between retrieval and language modules). We aim to carry out a similar investigation, but for the significantly less-studied task of issue priority prediction.

4 Approach

As described in Section 5.1, each item in our dataset of issues from public Jira databases consisted of a dictionary of information about the issue, with fields `ID`, `Summary`, `Project`, `Type` (for example Bug or Improvement), and `Description`, along with a ground-truth priority level (or label). These labels were assigned to the issue by the professional software engineers who created it, presumably with effort and internal discussions in order to maintain consistency with organizational guidelines. After reducing and filtering the original collection of issues to a manageable set of valid samples, we were left with 2500 training and 500 test examples, evenly distributed across priority levels.

To facilitate our priority prediction experiments on this dataset we built an evaluation framework, *FlowTest*, where we iterate over issue samples from our test set, run FM inference, and calculate priority prediction accuracy (as described below in Section 5.2). As our evaluation metric we used an exact match between the ground-truth label and the predicted level in the FM query response string (identified as all text between the `<priority>` tags). Note, malformed responses, for example due to missing tags, were treated as failed predictions. However, malformed responses only occurred in Experiment 1. With this infrastructure, we prepared the following two rounds of experiments.

4.1 Experimental Approach

Experiments 1–3: Experiment 1 uses GPT-3.5 Turbo as the baseline for comparison with GPT-3.5 Turbo + RAG, GPT-3.5 Turbo + FT, and GPT-3.5 Turbo + FT + RAG, as shown in Table 2. Each FlowTest sample is converted into a formatted string representation and concatenated with the prompt text, then fed to the FM via the OpenAI API. Predictions are extracted from the responses and compared with the ground truth labels. The procedure is then repeated in Experiments 2–3 using GPT-4 Turbo and GPT-4o, serving as baselines for GPT-4 Turbo + RAG and GPT-4o + RAG.

Experiments 4–6. For retrieval, we embedded text representations (paired with ground-truth priority) using OpenAI's *text-embedding-3-large* model into a vector database. During the model's forward pass, we embed the input issue sample and evaluate the cosine similarity with pre-calculated embeddings. The highest similarity vector is selected, and the corresponding text representation and label are inserted into the prompt for in-context learning. The procedure is then repeated with GPT-4 Turbo and GPT-4o for the GPT-4 Turbo + RAG and GPT-4o + RAG benchmarks.

Experiments 7–8. The training set is reformatted for the OpenAI API fine-tuning framework, and a job is initiated using a prediction loss similar to our evaluation metric: an exact match between a predicted priority string within `<priority>` tags. The resulting model weights are then evaluated. Experiment 8 then combines fine-tuned weights with retrieval-augmented in-context learning.

5 Experiments

We began with the hypothesis that retrieval-augmented in-context learning and fine-tuning would improve the performance of vanilla foundation models (FMs) in predicting the priority level of an issue in issue tracking systems (ITSs). As described in Section 4, we first used a dataset of issues to develop a novel evaluation, which we termed FlowTest, specifically for this task. We then evaluated vanilla FMs, as well as FMs enhanced with retrieval-augmented in-context learning and fine-tuning, using FlowTest. The dataset and experiments are detailed in the following subsections.

5.1 Data

We used a dataset of issues from Jira ITSs from Montgomery et al. (2022), selecting a subset of the fields shown in Figure 1 to ensure that suggestions were based solely on the task itself, excluding factors such as later comments or the name of the assignee. We then masked the priority of the issues and tasked the model configurations shown in Table 2 with predicting the priority level.

The inputs for the model configurations that we evaluated, as illustrated in Figure 4, are JSON representations of the issues. The output is the predicted priority level, which is added in a comment in Figure 4 below the JSON representation. The priority levels are Blocker, Critical, Major, Minor, and Trivial.

5.2 Evaluation Method

Our goal is to measure how effectively a model configuration can predict the priority level of an issue. This capability would enable the model to suggest priority levels in issue tracking systems, thereby helping to streamline task management.

We developed a novel evaluation method called FlowTest by sampling 500 issues from the Jira dataset from Montgomery et al. (2022). These issues were structured with the relevant information for prediction, and the priority levels were masked. We then evaluated whether a model configuration correctly predicted the priority level, assigning a score of 0 or 1 for each issue. The accuracy of the model configuration was calculated as the average score across the 500 issues in FlowTest.

We believe this is a robust dataset because it reflects hundreds of human judgments, often requiring consensus-building among different people working together. Our goal is to predict these human judgments ahead of time, thereby streamlining the coordination and decision-making process.

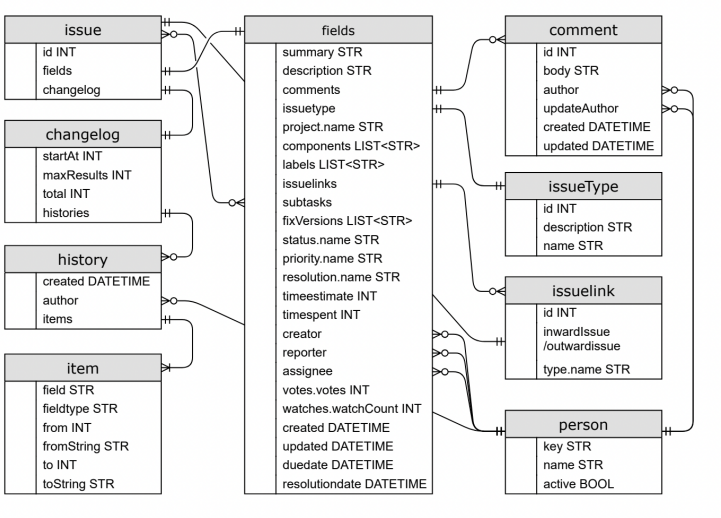

Figure 1: Jira database scheme from Montgomery et al. (2022).

5.3 Experimental Details

We conducted three rounds of experiments. Initially, we evaluated the vanilla FMs GPT-3.5 Turbo, GPT-4 Turbo, and GPT-4o. Next, we employed the *text-embedding-3-large* model to generate a vector database of 2,500 issues, implementing retrieval-augmented in-context learning with the same FMs. In the final round, we fine-tuned the GPT-3.5 Turbo FM and assessed its performance both with and without retrieval-augmented in-context learning.

Experiments 1–3. This stage involved extensive prompt engineering to maximize model performance. We employed JSON and tagging techniques to ensure consistent prediction formatting, enabling reliable extraction from model outputs. Most of the prompt engineering was focused on GPT-3.5 Turbo, and we observed that the same prompt generalized effectively to the larger foundation models, maintaining performance levels.

Experiments 4–6. In this stage, we created a vector database of $2,500$ issues using the *text-embedding-3-small* model. For each new call, we retrieved the most similar issue based on cosine similarity between the embeddings. While the embedding process was performed without masking the priority level, the input issue information used to query the database didn't include priority. We passed the resulting most relevant issue into the context of the FM, expecting that one-shot in-context learning would enhance FM performance. We used one-shot learning because of the limitation of FM context length.

Experiments 7–8. In this stage, we used a training set of $2,500$ issues from the Jira dataset, masking the priority level, and transforming the issues into the same format as the FlowTest evaluation. We then fine-tuned GPT-3.5 Turbo on these examples. Table 1 shows the fine-tuning time for the model, while Figure 2 illustrates the training loss for fine-tuning across the epochs, demonstrating smooth convergence. Finally, we applied the same retrieval process as before to the fine-tuned model.

Model Configurations	Training Time (HH:MM:SS)
GPT-3.5 Turbo + FT	01:24:35
GPT-3.5 Turbo + FT + RAG	01:24:35

Table 1: Training Time

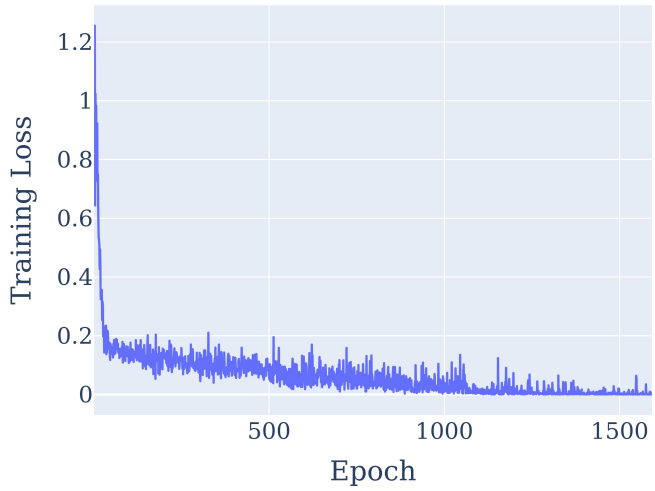

Figure 2: Training loss from fine-tuning GPT-3.5 Turbo.

5.4 Results

We found a TestFlow accuracy of 21.7%, 28.2%, and 29.6% for GPT-3.5 Turbo, GPT-4 Turbo, and GPT-4o, respectively, compared to a random sampling performance of 14.3%. With retrieval-augmented in-context learning, GPT-3.5 Turbo, GPT-4 Turbo, and GPT-4o achieved relative gains of 1.2%, 3.0%, and 7.4% over the vanilla FMs. The fine-tuned GPT-3.5 Turbo FM and the fine-tuned model with retrieval achieved relative gains of 32.0% and 33.2% over the accuracy of GPT-3.5 Turbo. The TestFlow accuracy and relative gains of the model configurations are shown in Table 2, while Figure 3 illustrates the TestFlow accuracy over the course of the evaluation for each model configuration on a scale of 0 to 1.

As illustrated, the larger FMs outperform the smaller ones on this task. Although all models perform better than random sampling, the TestFlow accuracy of GPT-3.5 Turbo is close to random, and GPT-4o is only correct about a third of the time. These results serve as our baselines to test if retrieval-augmented in-context learning and fine-tuning can improve performance, as hypothesized. Indeed, retrieval-augmented in-context learning results in relative gains of 1.2%, 3.0%, and 7.4% over each of the vanilla FMs. Notably, the gains are larger for the more advanced FMs, showing their greater ability to generalize in one-shot in-context learning.

We also find that fine-tuned GPT-3.5 Turbo significantly outperforms the vanilla GPT-3.5 Turbo, achieving a relative gain of 32.0%. It also surpasses all the model configurations with retrieval but without fine-tuning, demonstrating the substantial benefits of fine-tuning for this task. Notably, the fine-tuned GPT-3.5 Turbo exhibits the same relative gains from adding retrieval as GPT-3.5 Turbo without fine-tuning. This shows that while the fine-tuned model has adapted to the task, its ability to generalize remains the same as the model without fine-tuning and does not benefit more from retrieval than GPT-3.5 Turbo does.

Our findings indicate that retrieval provides a modest boost, particularly for larger models with strong generalization capabilities, while fine-tuning offers significant improvements, likely due to better local knowledge about the task. We expected retrieval to have a larger effect, but it appears that a single relevant example does not suffice to fully capture the context, necessitating more local knowledge. This suggests that as models scale, the effectiveness of retrieval will improve, but fine-tuning remains crucial for this dataset due to its inherent complexity.

GPT-3.5 Turbo + FT + RAG achieved an accuracy of 0.549%, which translates to more than a 50% chance of making the correct prediction. This is impressive given the difficulty humans face in distinguishing between Minor and Trivial or Major and Critical tasks. We believe that fine-tuning GPT-4 Turbo or GPT-4o could yield even better results. There is further discussion in Section 6.

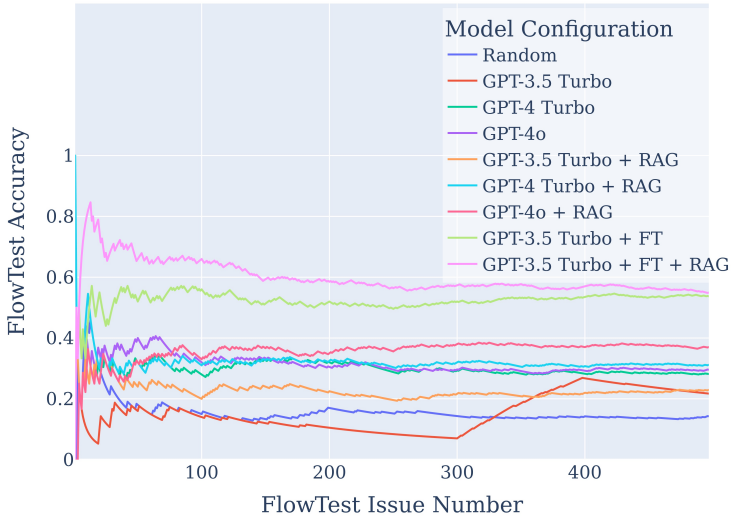

Figure 3: FlowTest accuracy of model configurations over the course of the evaluation.

Model Configurations	FlowTest Accuracy	Relative Gains
Random	14.3%	–
GPT-3.5 Turbo	21.7%	–
GPT-4 Turbo	28.2%	–
GPT-4o	29.6%	–
GPT-3.5 Turbo + RAG	22.9%	1.2%
GPT-4 Turbo + RAG	31.2%	3.0%
GPT-4o + RAG	37.0%	7.4%
GPT-3.5 Turbo + FT	53.7%	32.0%
GPT-3.5 Turbo + FT + RAG	54.9%	33.2%

Table 2: FlowTest accuracy and relative gains from RAG + FT of model configurations.

6 Analysis

In this section, we conduct a qualitative evaluation using chain-of-thought outputs from the model configurations to identify the sources of performance improvement resulting from retrieval-augmented in-context learning. By examining examples where GPT-4o and GPT-4o + RAG reasoned similarly, we observed that the latter benefited from referencing a similar issue within the same project. This additional context enabled better calibration of assigned priority levels, as having a reference issue helps determine whether the priority level should be lower, the same, or higher.

We further analyze model configuration predictions, identifying consistent correctness and errors in some samples. Two examples are shown in Figure 4. The model configurations consistently successfully predict Issue 1 due to the clear task description, which does not require extensive project knowledge to assess the priority level. Conversely, Issue 2 demands additional project context to understand the importance of *MonitorMemory*. The models need to know if MonitorMemory is a critical component of the project or merely a logging tool. Lacking this context, the models tend to predict a higher priority level than is warranted.

```
 // Issue 1

{'Issue ID': '13052868',
 'Issue Summary': 'Make multiple COUNT(DISTINCT) message state workarounds',
 'Project': 'IMPALA',
 'Issue Type': 'Improvement',
 'Description': 'The message for a query with multiple COUNT(DISTINCT) '
                'currently looks like:\n'
                '\n'
                '[localhost:21000] > select count(distinct x), count(distinct '
                'y) from t1;\n'
                'ERROR: AnalysisException: all DISTINCT aggregate functions '
                'need to have the same set of parameters as count(DISTINCT x); '
                'deviating function: count(DISTINCT y)\n'
                '\n'
                'How about adding an INFO message after the error saying '
                'something like:\n'
                '\n'
                'INFO: If estimated counts are OK, replace COUNT(DISTINCT) '
                'with NDV(). Enable the APPX_COUNT_DISTINCT query option to '
                'perform this rewrite automatically.'}

// Priority Level: Minor

// Issue 2

{'Issue ID': '13037067',
 'Issue Summary': 'MonitorMemory produces UnsupportedOperationException',
 'Project': 'Apache NiFi',
 'Issue Type': 'Bug',
 'Description': '{code}\n'
                '[na:1.8.0_111] Caused by: '
                'java.lang.UnsupportedOperationException: Usage threshold is '
                'not supported at '
                'sun.management.MemoryPoolImpl.setUsageThreshold(MemoryPoolImpl.java:114)
                    '
                '~[na:1.8.0_111] at '
                'org.apache.nifi.controller.MonitorMemory.onConfigured(MonitorMemory.java:178)
                    '
                '~[na:na] ... 16 common frames omitted\n'
                '{code}\n'
                '\n'
                'MonitoryMemory checks if '
                'memoryPoolBean.isCollectionUsageThresholdSupported()) and if '
                'so it eventually calls '
                'monitoredBean.setUsageThreshold(calculatedThreshold), but '
                'setUsageThreshold then checks isUsageThresholdSupported() so '
                'it can throw UnsupportedOperationException. '}

// Priority Level: Minor
```

Figure 4: Jira issues from the FlowTest evaluation.

Through chain-of-thought prompting, we discern differences in model reasoning. Notably, GPT-4o employs a more consistent reasoning format compared to GPT-3.5 Turbo and GPT-4 Turbo, and demonstrates greater confidence in its predictions. These insights underscore the importance of both retrieval-augmented in-context learning and fine-tuning. While retrieval provides valuable contextual references that enhance model predictions, fine-tuning offers significant performance improvements by adapting the model to the specific nuances of the dataset. Our analysis indicates that combining these approaches effectively enhances model accuracy and reliability in predicting issue priorities.

7 Conclusion

Our hypothesis that fine-tuning and retrieval using a relevant ITS database would significantly improve issue priority prediction over vanilla FM baselines has been validated. We introduced a novel dataset and evaluation framework, termed *FlowTest*, for ITS priority prediction, based on the public ITS repositories from Montgomery et al. (2022). With retrieval, GPT-3.5 Turbo, GPT-4 Turbo, and GPT-4o achieved relative gains of 1.2%, 3.0%, and 7.4% respectively over the vanilla FMs. The fine-tuned GPT-3.5 Turbo and the fine-tuned model with retrieval achieved substantial relative gains of 32.0% and 33.2% over the baseline GPT-3.5 Turbo.

Our findings indicate that retrieval provides a modest boost, particularly for larger models with strong generalization capabilities. However, fine-tuning offers significant improvements, likely due to enhanced local knowledge about the task. Although we anticipated a more substantial effect from retrieval, it appears that a single relevant example does not fully capture the necessary context, underscoring the importance of fine-tuning for this complex dataset. As models scale, we expect retrieval effectiveness to improve, yet fine-tuning remains crucial. Notably, the highest accuracy of 0.549% achieved by GPT-3.5 Turbo + FT + RAG is impressive given the difficulty humans face in distinguishing between task priorities such as Minor versus Trivial and Major versus Critical.

Qualitative evaluations using chain-of-thought outputs revealed differences in reasoning between the models. While retrieval provides valuable contextual references that enhance model predictions, fine-tuning significantly improves performance by adapting the model to the specific nuances of the dataset. Our analysis demonstrates that combining these approaches effectively enhances model accuracy and reliability in predicting issue priorities, suggesting substantial utility for real-world software teams seeking to streamline their ITS processes.

The limitations of the project include a relatively small and less curated dataset of 500 samples in FlowTest, which is sometimes difficult for humans to interpret. The vector database of $2,500$ issues, while useful, could have been expanded given the $1,000,000$ issues in the Apache dataset, to provide more similar issues for one-shot in-context learning. Future work should focus on refining the evaluation and exploring advanced retrieval-augmented in-context learning techniques, such as DSPy, at higher costs to further increase accuracy. Additionally, developing a tool that predicts issue priorities for task management could be highly beneficial for organizations.

8 Ethics Statement

Challenge 1. When using model configurations to assign priority levels to tasks within organizations, there is a possibility that these models will inherit biases from the pre-training and fine-tuning stages. While identifying information has been removed, it is possible that the model configurations have learned biases based on writing style. Before deploying the model configurations in organizations, it is advisable to audit the pre-training and fine-tuning datasets and monitor the model outputs for bias.

Challenge 2. While the FlowTest accuracy of GPT-3.5 Turbo + FT + RAG is 54.9%, there is a risk of over-reliance on these models for assigning priority levels to tasks within organizations. Before deployment, it is important to communicate that these models are tools to support rather than replace decision-making. Before deploying model configurations in organizations, it is advisable to prompt the models to provide a rationale for their suggestions, to promote transparency and accountability.

References

Google DeepMind. 2024. Gemini: A family of highly capable multimodal models.

Jayati Deshmukh, KM Annervaz, Sanjay Podder, Shubhashis Sengupta, and Neville Dubash. 2017. Towards accurate duplicate bug retrieval using deep learning techniques. In *2017 IEEE International conference on software maintenance and evolution (ICSME)*, pages 115–124. IEEE.

Jianjun He, Ling Xu, Meng Yan, Xin Xia, and Yan Lei. 2020. Duplicate bug report detection using dual-channel convolutional neural networks. In *Proceedings of the 28th International Conference on Program Comprehension*, pages 117–127.

Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. 2009. Improving bug triage with bug tossing graphs. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 111–120.

Omar Khattab, Christopher Potts, and Matei Zaharia. 2021. Relevance-guided supervision for openqa with colbert.

Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. 2023. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp.

Ahmed Lamkanfi, Serge Demeyer, Emanuel Giger, and Bart Goethals. 2010. Predicting the severity of a reported bug. In *2010 7th IEEE working conference on mining software repositories (MSR 2010)*, pages 1–10. IEEE.

Ahmed Lamkanfi, Serge Demeyer, Quinten David Soetens, and Tim Verdonck. 2011. Comparing mining algorithms for predicting the severity of a reported bug. In *2011 15th European Conference on Software Maintenance and Reengineering*, pages 249–258. IEEE.

Clara Marie Lüders, Abir Bouraffa, and Walid Maalej. 2022. Beyond duplicates: towards understanding and predicting link types in issue tracking systems. In *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR '22. ACM.

Lloyd Montgomery, Clara Lüders, and Walid Maalej. 2022. An alternative issue tracking dataset of public jira repositories. In *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR '22. ACM.

OpenAI. 2020. Language models are few-shot learners.

OpenAI. 2024. Gpt-4 technical report.

Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. 2008. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th international conference on Software engineering*, pages 461–470.