Artificial Estimators

A Study in Training Systems for Expert Estimating Software Development Task Efforts

A Senior Honors Project Presented to the Faculty of the Department of Information & Computer Science, University of Hawaiʻi at Mānoa

In Partial Fulfillment of the Requirements for the Bachelor of Computer Science with Honors

By Andrea W Jans
November 10, 2025

Committee:
Dr. Daniel Port, Mentor
Dr. Anthony Peruma
Dr. Italo de Oliveira Santos

Abstract

Effort estimation in software engineering has long been a persistent challenge, with traditional expert-based and algorithmic approaches often producing inaccurate and inconsistent results. This research investigates whether large language models (LLMs), specifically when combined with Generative Pretrained Transformers (GPT) and Retrieval-Augmented Generation (RAG), can provide reliable estimates of the effort required for software development tasks. A mixed-methods research design was used: historical project data was collected, standardized, and used to fine-tune an open-source LLM, which was then tested against untrained configurations, with and without RAG. Thirty feature requests were used as test cases in multiple runs to evaluate performance. The results indicate that while LLMs can generate effort estimates, their outputs are highly inconsistent. The research suggests that, despite their potential, current LLM-based systems cannot yet replace expert judgment in estimating software projects. Instead, they highlight the need for further refinement, the integration of domain-specific training, and standardized data practices to harness LLMs as supportive—rather than standalone—tools for estimating software engineering efforts.

Keywords: Artificial Intelligence, Software Engineering, Estimation

Acknowledgments

I am deeply grateful to Dr. Daniel Port for his mentorship and unwavering belief in me. I also extend my sincere thanks to Dr. Anthony Peruma for his steadfast support throughout this process, and to Dr. Italo de Oliveira Santos for his invaluable feedback.

My special thanks go to my wife and family, without whom this journey would not have been possible.

The technical support and advanced computing resources from University of Hawaiʻi Information Technology Services – Research Cyberinfrastructure, funded in part by the National Science Foundation CC* awards #2201428 and #2232862, are gratefully acknowledged.

Table of Contents

Table of Figures

Introduction

In computer science, a common task often delegated to experts is to estimate the effort required to fix a programming defect or develop a new feature. Based on rules of thumb and intuition, efforts are made to evaluate the reliability of expert estimates. Feature enhancement estimations have traditionally been more complicated, as there are no algorithmic or empirical models to produce reliable estimates.

The NASA Jet Propulsion Laboratory (JPL) MONTE team is frequently asked to add new features to its software library. Currently, they rely on expert estimators to evaluate the effort and cost for each addition. Using empirically derived heuristics, these estimators target an 80% reliability level, helping project managers assure internal stakeholders that budget overruns will be rare or within an expected range. The level of confidence needs to be reliable, meaning that 80% of the time, the actual effort will indeed be equal to or less than the fair estimate. Unfortunately, we have no principled basis that expert estimation is reliable in this way, but empirical measurements over time have shown this to be true for the MONTE team. These measurements show that the error follows an exponential trend as the feature request becomes more complex. Could a system based on a fine-tuned Large Language Model (LLM), powered by a Generative Pretrained Transformer (GPT), be trained using historical estimation data to reliably predict the feature enhancement effort required at a satisfactory reliability level?

Project Significance

While Lavazza et al. [6] used traditional machine learning techniques for software effort estimation instead of leveraging LLMs, this study explores the potential of generative artificial intelligence (AI). Specifically, it is among the first to investigate applying a GPT-based LLM enhanced with Retrieval-Augmented Generation (RAG) to produce reliability-informed estimates for software engineering tasks. By combining GPT architectures with RAG mechanisms, this research aims to determine whether such models can deliver more consistent estimates than traditional machine learning methods. This represents a new direction in effort estimation research, bridging the gap between data-driven predictive techniques and recent advances in natural language understanding. As an exploratory research project, the goal was to expand the knowledge base regarding the application of machine learning and LLMs in software engineering effort estimation.

Research Questions

- **RQ1**: Can an LLM provide consistent estimates that fall within an 80% reliability interval?
    - **RQ1.1**: Can providing targeted information via RAG allow an LLM to provide consistent estimates that fall within an 80% reliability interval?
- **RQ2**: Can an LLM with additional fine-tuning on feature requests provide consistent estimates that fall within an 80% reliability interval?
    - **RQ2.1**: Can providing targeted information via RAG allow a fine-tuned LLM to provide consistent estimates that fall within an 80% reliability interval?

Project Contributions

This study presents a new application of LLMs and RAG for effort estimation and provides experimental evidence of their current limitations. It also offers guidelines for future AI–human collaboration in software project estimation. Additionally, the study advocates for community data standards and sharing to improve AI reliability in engineering fields.

Literature Review

Effort Estimation

Effort estimation in software development has been a persistent challenge since the 1960s, prompting extensive research and exploration into potential solutions. Halkjelsvik and Jørgensen [4] shed light on the prevailing issues in this domain, revealing that deviations exceeding one-third of the initial estimate were common. Moreover, these deviations predominantly led to increased actual effort relative to the initially estimated effort.

Jørgensen [5] surveyed software engineering estimation methods. Finding a standard set of "best practices" for software engineering estimation by experts, Jørgensen [5] laid out 12 guidelines:

1. Evaluate estimation accuracy, but avoid high evaluation pressure
2. Avoid conflicting estimation goals
3. Ask the estimators to justify and criticize their estimates
4. Avoid irrelevant and unreliable estimation information
5. Use documented data from previous development tasks
6. Find estimation experts with a relevant domain background and good estimation records
7. Estimate top-down and bottom-up, independently of each other
8. Use estimation checklists
9. Combine estimates from different experts and estimation strategies
10. Assess the uncertainty of the estimate
11. Provide feedback on estimation accuracy and development task relations
12. Provide estimation training opportunities.

The empirical research findings of Halkjelsvik and Jørgensen [4] underline a pervasive trend in software project management: estimates often fall short of accurately predicting the required effort. These discrepancies can lead to various challenges, including budget overruns, missed deadlines, and compromised project quality.

Past Solutions

The COCOMO model proposed by Boehm [1] emphasizes the intricacies of estimating software development effort and highlights the qualitative aspects of the process. The COCOMO model [2] provides a systematic way to calculate costs and effort, identifies the factors with the most significant impact on price and effort, and assesses the feasibility of a software project. However, the COCOMO model [2] assumes that software size is the primary factor, does not account for the specific characteristics of the software engineering team, and lacks the precision it could achieve. Also, because it focuses on the entire project, it cannot be used to address individual feature requests during a project's maintenance phase.

According to Shepperd et al. [8], one significant challenge to the reliability of expert estimates is bias in the estimator's judgment. Biases can skew the estimation process, leading to flawed decisions and outcomes because they cannot be fully quantified. One common bias observed in expert estimates is the anchoring bias. Initial information, often called an anchor, may not be entirely accurate or relevant, which introduces bias. Despite subsequent corrections or additional information, the estimator tends to cling to the original anchor, even when it is incorrect. This results in distortions in judgment and estimation [8].

Dagnino [3] introduced the ONSET (One Software Estimation Toolkit), a method for estimation in situations where historical data is lacking. ONSET amalgamates concepts, principles, and techniques from the software estimation domain that are typically treated as separate entities in the literature. This toolkit is particularly valuable for organizations facing unreliable historical data. The paper illustrates the application of these principles and methods within ONSET, demonstrating how estimates can be derived effectively without relying on intricate or costly tools.

Machine Learning

In a current survey of software engineering effort estimation techniques, Saeed et al. [7] reiterated that effort estimation in software development is crucial, as it significantly impacts project planning, resource allocation, and overall project success. One key takeaway emphasized by the research is that different effort estimation models have advantages and limitations. These models are not one-size-fits-all solutions but tools tailored to specific project requirements and conditions. The choice of estimation model depends on factors such as project size, complexity, team composition, and the availability of historical data. The survey did find that supervised learning algorithms are the most popular for effort estimation.

Lavazza et al. [6] applied machine learning techniques to estimate software functional size early. Their objective was to investigate whether machine learning methods could produce more precise estimates of Functional Point Analysis (FPA) measures compared to traditional approaches, such as High-level FPA. They conducted an empirical study using a substantial dataset of functional size predictors to train and assess three widely recognized and robust machine learning algorithms: Random Forests, Support Vector Regression, and Neural Networks. The study involved a systematic experimental process encompassing dataset filtering, splitting, parameter optimization, model training, and validation cycles. Following this methodology, Lavazza et al. [6] evaluated the estimation accuracy of the machine learning models and compared them against fixed-weight models (e.g., High-level FPA) and linear regression models.

Additionally, the performance of these models was evaluated using a separate dataset as a test set to validate their effectiveness. The findings suggest that Support Vector Regression produces relatively accurate estimation models. However, the level of accuracy achieved is similar to that of High-level FPA or models constructed using ordinary least squares regression. Lavazza et al.

[6] observed that models capable of generating accurate estimates often require differentiation only between transaction and data types.

The groundwork has been laid for the next generation of machine learning algorithms, including LLMs, to perform software engineering effort estimation. The LLM finds similar tasks across the training data using multiple methodologies and historical data. This historical data, based on functional specifications, estimates, and actual effort, should provide accurate estimations with high reliability.

Research Methods

The methodology used in this study followed a mixed-methods, experimental design to assess the consistency and reliability of LLMs in software development effort estimation. As shown in the research flow (Figure 1), the process consisted of four distinct phases: Design, Transform, Experiment, and Measure. The core of the investigation involved using a Free and Open-Source Software (FOSS) LLM, specifically Llama-3.2-3B-Instruct, which was tested under various conditions. Historical project data, consisting of 1161 documents from the NASA JPL MONTE team's software library, were first standardized and converted into two usable formats: a vector database for RAG and a fine-tuning dataset. Four sequential experiments—Direct, Direct RAG, Trained, and Trained RAG—were then performed to systematically evaluate the impact of fine-tuning and contextual grounding on the model's ability to produce estimates for a set of 30 feature request prompts (see Appendix B), with the primary measurement being the consistency of the output within an 80% reliability interval.

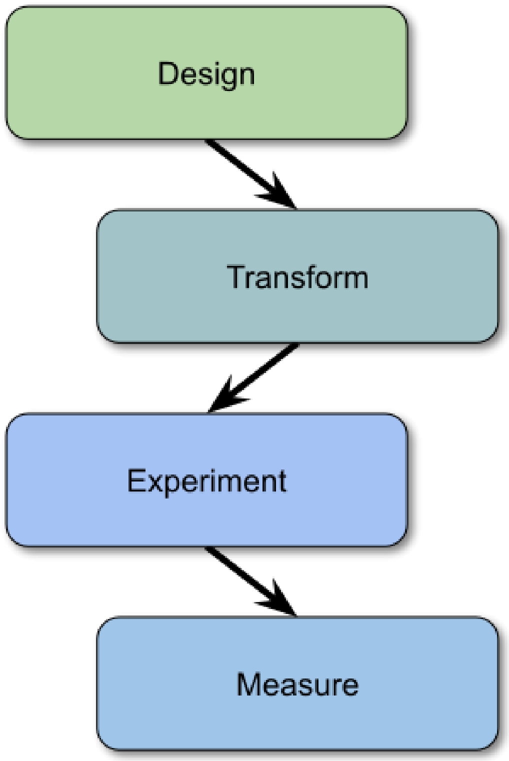

Figure 1: Research Methods

Design

Since this research project did not focus on building a machine learning system with an LLM from scratch, the study identified a free and open-source software (FOSS) LLM model (Llama-3.2-3B-Instruct). The information recorded by the MONTE project team (see Appendix A) was transformed and stored in a vector database for RAG and used to fine-tune a copy of the LLM. The strength of the data source is that it is actual field data collected by an organization interested in improving the software engineering effort estimates used to model the cost and schedule estimates of future software projects. The weakness is that the dataset contained only 1161 documents, excluding the export restrictions notification.

Transform

The data needed to be transformed into a standard format for digestion by the vector database and for fine-tuning. To create the documents stored in the vector database, the description and comments were concatenated into the document body. The enhancement ID, worktime, product, and component were stored as metadata.

To fine-tune the LLM, the documents were passed with the following instruction: "Estimate the time in hours to complete the feature request." The question was the body of the document. The answer was formatted as "The feature will take approximately {+ str(document.meta['Work-Time']) +' hours}' to complete." The double curly braces in the answer were intended to help train the LLM so that the final estimate would be easy to parse from the text response. The training data consisted of the first 80% of the documents, with the remainder reserved for testing the fine-tuning process. The research project used the KOA High Performance Cluster to virtualize computing resources during fine-tuning of the Llama-3.2-3B-Instruct LLM, thereby avoiding the need for physical hardware investment.

Experiment

The first experiment (Direct) aimed to answer RQ1. The prompts used were the 30 feature request prompts (see Appendix B), along with follow-up instructions to make extracting the answer easier: "Always give the estimate in hours, and always enclose only the final estimate in double curly braces." The second experiment (Direct RAG) aimed to answer RQ1.1. Additional context from searching the vector database for the top five documents related to the feature request was added to the prompt. The same follow-up instructions were provided. The third experiment (Trained) aimed to answer RQ2. The prompts consisted solely of the feature requests and instructions to estimate in hours, and to enclose the final estimate in double curly braces. The fourth experiment (Trained RAG) aimed to answer RQ2.1. Additional context from searching the vector database for the top five documents related to the feature request was added to the prompt, with the same formatting instructions for the answer.

Measure

The primary measurement was the estimates produced by the LLMs based on the 30 feature request prompts (see Appendix B) in each experiment, across the ten passes. Each experiment was executed sequentially, from first to fourth, in a loop, with a count from 1 to 10. This produced 10 data sets for each experiment. The datasets were indexed by experiment (Direct, Direct RAG, Trained, Trained RAG) and by feature request, yielding 1200 data points. The instructions for identifying the final estimate were inconsistently followed, resulting in a significant number of 'N/A' responses, which were recorded as zeros.

Results

This section presents the findings from the four sequential experiments designed to answer the central research questions (RQ1, RQ1.1, RQ2, and RQ2.1). The experimental process generated a total of 1,200 data points, derived from 30 feature request prompts (see Appendix B) run across 10 passes for each of the four LLM configurations (Direct, Direct RAG, Trained, and Trained RAG). The primary analytical focus was to determine whether the LLM's estimates consistently fell within a desired 80% reliability interval, reflecting the reliability target set by expert estimators. Initial data cleaning was required because the LLM inconsistently adhered to the requested output format, leading to several 'N/A' responses that were subsequently recorded as zeros. The results are presented in four corresponding subsections, each using scatterplots and candlestick plots to visually represent the distribution, range, and variability of the time estimates (in hours) across the test cases.

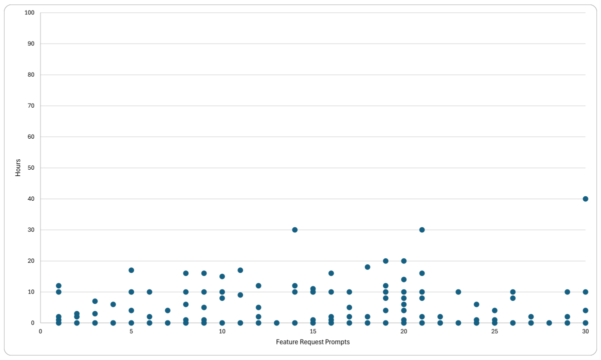

Figure 2: Direct Experiment Scatterplot

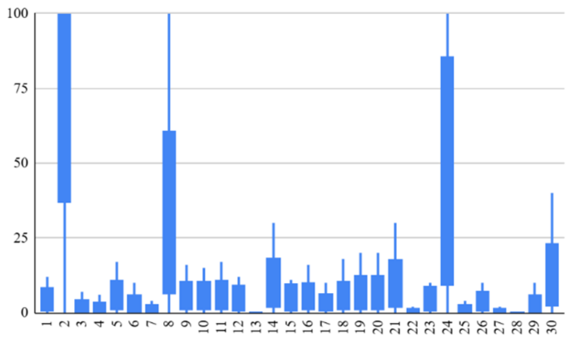

Figure 3: Direct Experiment Candlestick Plot

RQ1: Can an LLM provide consistent estimates that fall within an 80% reliability interval?

Figure 2 presents a scatterplot of LLM time estimates (in hours) across the thirty prompts given to the LLM. The distribution reveals that most estimates clustered below 25 hours, indicating generally low perceived task duration. However, several prompts—particularly prompts 2, 8, and 23—elicited substantially higher estimates, reaching over 100 hours. Figure 3 provides a complementary candlestick representation of the same data, summarizing the variability in responses for each prompt with high, low, and an 80% reliability factor. Some prompts exhibit narrow candlesticks with short wicks, reflecting consistent estimates across execution runs, whereas prompts 2, 8, 14, 21, 24, and 30 show extended ranges, highlighting considerable disagreement or uncertainty. As the number of prompts showing disagreement and uncertainty is significant — 1 out of every 5 — the LLM fails to meet the 80% reliability threshold, and the answer to RQ1 must be "No."

RQ1.1: Can providing targeted information via RAG allow an LLM to provide consistent estimates that fall within an 80% reliability interval?

Figure 4 presents a scatterplot of LLM with RAG time estimates (in hours) across the thirty prompts given to the LLM. The distribution shows that most estimates cluster below 25 hours, indicating generally low perceived task duration. However, several prompts—particularly prompts 4, 6, 12, 23, and 29—elicited significantly higher estimates, exceeding 100 hours. Figure 5 provides a complementary candlestick chart of the same data, summarizing response variability for each question with high, low, and an 80% reliability interval. Most prompts have narrow candlesticks, reflecting consistent estimates across runs, while prompts 4, 6, 10, 12, 13, 14, 20, 23, 25,26, 29, and

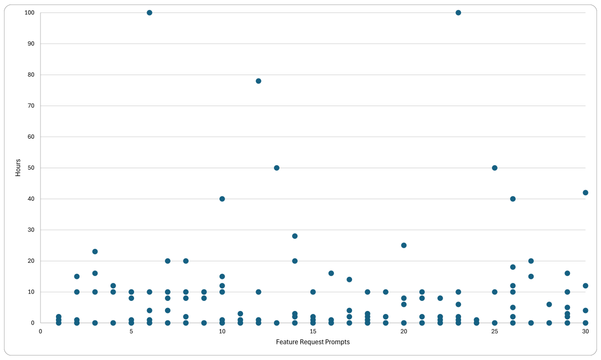

Figure 4: Direct RAG Experiment Scatterplot

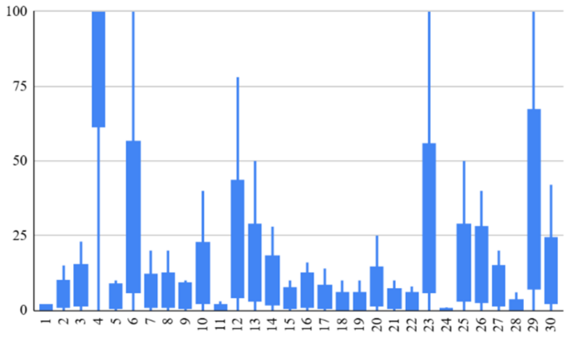

Figure 5: Direct RAG Experiment Candlestick Plot

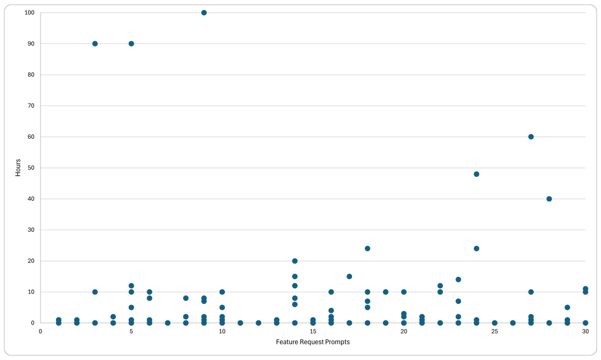

Figure 6: Trained Experiment Scatterplot

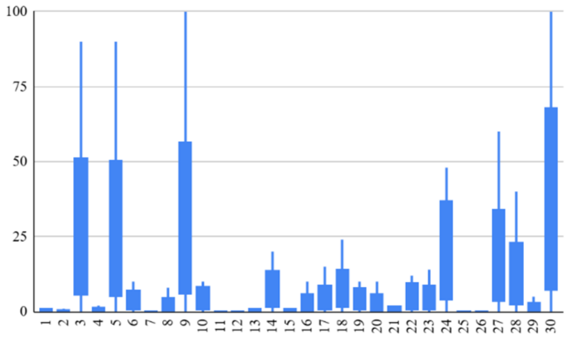

Figure 7: Trained Experiment Candlestick Plot

30 display wider ranges, indicating considerable disagreement or uncertainty. As the number of prompts showing disagreement and uncertainty is significant — 2 out of every 5 — the LLM fails to meet the 80% reliability threshold, and the answer to RQ1.1 must be "No."

RQ2: Can an LLM with additional fine-tuning on feature requests provide consistent estimates that fall within an 80% reliability interval?

Figure 6 presents a scatterplot of LLM with RAG time estimates (in hours) across the thirty prompts given to the LLM. The distribution shows that most estimates cluster below 25 hours, in-

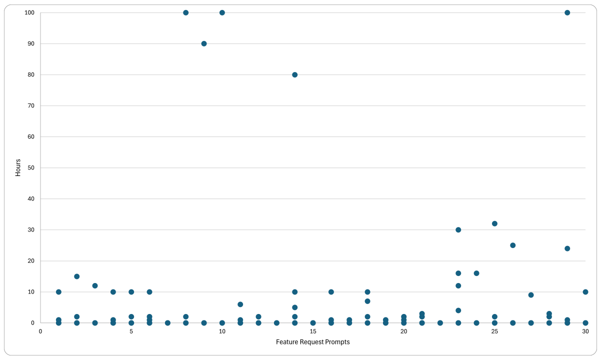

Figure 8: Trained RAG Experiment Scatterplot

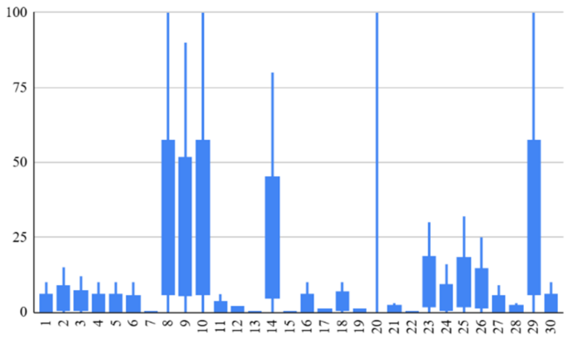

Figure 9: Trained RAG Experiment Candlestick Plot

dicating generally low perceived task duration. However, several prompts—particularly prompts 3, 5, 9, 27, and 30—elicited significantly higher estimates, exceeding 100 hours. Figure 7 provides a complementary candlestick chart of the same data, summarizing the response variability for each question, highlighting high and low values, and showing an 80% reliability interval. Most prompts have narrow candlesticks, reflecting consistent estimates across runs, while prompts 3, 5, 9, 24, 27, 28, and 30 display wider ranges, indicating considerable disagreement or uncertainty. As the number of prompts showing disagreement and uncertainty is significant — approximately 1 out of every 5 — the LLM fails to meet the 80% reliability threshold, and the answer to RQ2 must be "No."

RQ2.1: Can providing targeted information via RAG allow a fine-tuned LLM to provide consistent estimates that fall within an 80% reliability interval?

Figure 8 presents a scatterplot of LLM with RAG time estimates (in hours) across the thirty prompts given to the LLM. The distribution shows that most estimates cluster below 25 hours, indicating generally low perceived task duration. However, several prompts—particularly prompts 8, 9, 10, 14, 20, and 29—elicited significantly higher estimates, exceeding 100 hours. Figure 9 provides a complementary candlestick chart of the same data, summarizing response variability for each question with high, low, and an 80% reliability interval. Most prompts have narrow candlesticks, reflecting consistent estimates across runs, while prompts 8, 9, 10, 14, 20, 23, 25, 26, and 29 display wider ranges, indicating considerable disagreement or uncertainty. As the number of prompts showing disagreement and uncertainty is significant — approximately 1 out of every 3 — the LLM fails to meet the 80% reliability threshold, and the answer to RQ2.1 must be "No."

Threats to Validity

The validity of this research on using LLMs for software effort estimation faces several challenges. Historical project data from the MONTE project team was used to fine-tune the LLM and RAG retrievers. Although it is real-world field data, the dataset's relatively small size (1161 documents) and the natural variability in how human experts documented estimates and actual effort could introduce noise and limit the LLM's learning. The structure of the Jira data (see Appendix A) indicates potential ambiguities, as the "Hours Worked" field shows total logged hours rather than the estimated effort.

The experiments relied on specific formatting instructions ("Always give the estimate in hours, and always enclose only the final estimate in double curly braces") to extract numerical estimates from the LLM's text response. The LLM's inconsistent adherence to these instructions led to a significant number of 'N/A' answers, which were recorded as zeros, potentially skewing the quantitative results (e.g., the mean and reliability intervals).

The study used a single, specific open-source LLM, Llama-3.2-3B-Instruct. The results are tied to this model's inherent architecture and training. Using a different base LLM or changing hyperparameters during fine-tuning can yield different results, making it hard to apply these findings across all LLMs.

The historical data and the 30 feature requests used as test cases all come from the NASA JPL MONTE team's software library. While this creates a consistent dataset, the domain-specific nature of the data limits the ability to generalize the conclusion—that LLMs cannot yet replace expert judgment—to software projects in entirely different fields.

The training dataset included only 1161 documents. This modest size for a machine learning project, especially for an LLM, limits the model's exposure to various project scenarios and the full range of estimation complexities. This limitation could be a primary reason for the observed inconsistency.

The research questions (RQ1, RQ1.1, RQ2, RQ2.1) focus on whether the estimates fall within an 80% reliability interval. While this is a standard metric for assessing reliability, it may not fully reflect the qualitative aspects of expert-level estimation, such as the reasoning or qualitative risk assessment that human experts use.

The study assessed consistency over ten passes for each of the four experiments. While this indicates run-to-run stability, consistency in software estimation also requires alignment with the actual effort needed. Since the real effort for the 30 upcoming feature requests is not provided, the "consistency" measure mostly shows the model's internal reproducibility rather than its ability to predict the actual effort.

Conclusions

This research aimed to investigate whether LLMs, when fine-tuned with historical software project data combined with GPT and RAG, can provide reliable effort estimations for software engineering tasks. The motivation stemmed from the long-standing challenges in software project management, where inaccurate estimates have often led to cost overruns, missed deadlines, and reduced quality.

The findings of this study demonstrate that current LLMs are unable to produce consistent or dependable estimates. Across multiple test configurations—direct use, fine-tuning, and integration with RAG—results showed significant variability, including frequent outliers, incomplete responses, and poor reproducibility across runs. Despite their impressive natural language capabilities and ability to emulate expert-like reasoning, LLMs still struggle to map software task descriptions to accurate estimates of effort consistently.

Several important conclusions emerge:

1. **LLMs cannot yet replace human expertise in effort estimation**. Human judgment, particularly when guided by established estimation frameworks and domain experience, remains more reliable.

2. **Data quality and standardization are critical.** variability in historical project data limited the effectiveness of training, underscoring the need for shared, structured repositories of estimation data within the software engineering community.

3. **LLMs show potential as supportive tools.** While they cannot serve as primary estimators, they may provide value as assistants—generating alternative perspectives, highlighting overlooked factors, or serving as part of a hybrid human–AI estimation workflow.

4. **Future research is necessary.** Progress will depend on the development of improved

fine-tuning methods, domain-specific training datasets, and the integration of LLMs into broader estimation frameworks, rather than treating them as standalone solutions.

In conclusion, although this study demonstrates that LLMs are not yet capable of achieving expert-level accuracy in software effort estimation, they remain a promising area for further research. With more rigorous training data, better alignment techniques, and thoughtful integration into existing estimation practices, LLMs may evolve into valuable complements to expert judgment rather than replacements.

Future Work

Building on the findings of this study, several directions for future research are proposed:

1. **Standardized Data Repositories** – Developing a shared, community-driven dataset of software project specifications, estimates, and actual outcomes would provide a stronger foundation for training and benchmarking AI-based estimators.

2. **Domain-Specific Fine-Tuning** – LLMs trained on generic text struggle to capture the nuances of software engineering estimation. Future research should focus on fine-tuning models using curated, domain-specific corpora to align them with estimation tasks better.

3. **Hybrid Human–AI Workflows** – Instead of replacing expert estimators, future systems should explore how AI can complement human judgment, for example, by providing alternative estimates, surfacing risks, or quantifying uncertainty ranges.

By pursuing these directions, the software engineering community can advance toward AI-assisted estimation systems that are robust, transparent, and truly valuable in practice.

Appendix A: Jira Data Structure

The top-level structure is a JSON object where keys are ID strings and values are record objects.		
Key	**Type**	**Description**
[ID_STRING]	Object	A single record, keyed by the record's primary ID. Example key: "123".

Record Object Fields		
The record object contains the main details of the issue.		
Field Name	**Type**	**Description**
Id	Number (Integer)	The unique numerical identifier for the record.
Opened	String	The timestamp for when the record was initially opened/created (e.g., "YYYY-MM-DD HH:MM:SS").
Description	String	A concise title or summary of the issue.
Submitted By	String	The email address or identifier of the user who submitted the record.
Last Modified	String	The timestamp for the last time any change or comment was recorded.
Product/Component	String	Identifies the software product and specific component this record relates to (e.g., "Product/Component").

Current Status	String	The current state of the record (e.g., “RESOLVED”, “CONFIRMED”, “NEW”).
Resolution	String	The outcome or method of resolution (e.g., “DELIVERED”, “FIXED”, “WONTFIX”).
Owner	String	The email address or identifier of the person currently responsible for the record.
Hours Worked	Number (Float)	The total number of hours logged as worked on this record.
Comments	Object	A nested object containing a history of all user comments.
Changes	Object	A nested object containing a history of all field changes.

Nested Object: Comments		
The Comments object stores the discussion timeline and the initial description.		
Key	**Type**	**Description**
[TIMESTAMP_STRING]	String	The key is the timestamp (e.g., “YYYY-MM-DD HH:MM:SS”) of when the comment was added. The value is the comment body, often prepended with the author’s email (e.g., “email:: comment text”).

Nested Object: Changes		
The Changes object stores a detailed history of modifications to the record’s fields.		
Key	**Type**	**Description**

[TIMESTAMP_STRING]	Object	The key is the timestamp (e.g., “YYYY-MM-DD HH:MM:SS”) of the change. The value is an object detailing the changes made at that time.

Changes Object Fields		
The object keyed by the timestamp contains at least a who field, and then one or more fields named after the field that was changed (e.g., bug_severity, priority).		
Field Name	**Type**	**Description**
who	String	The email address or identifier of the user who made the change.
[CHANGED_FIELD_NAME]	Object	A nested object detailing the specific field change. Examples: bug_severity, resolution, work_time.

Changed Field Detail Object Fields		
Field Name	**Type**	**Description**
from	String	The value of the field before the change.
to	String	The new value of the field after the change.

Appendix B: 30 Feature Request Prompts

1. “Estimate the effort for adding resizing the plot when you size the window to the system.”
2. “Estimate the effort for adding trajectory alternatives to the system.”
3. “Estimate the effort for adding a new type of plot to the system.”
4. “Estimate the effort for adding the Offset Trajectory to the system.”
5. “Estimate the effort for adding the ability to change the color of the plot to the system.”
6. “Estimate the effort for adding the ability to change the line style of the plot to the system.”
7. “Estimate the effort for adding the ability to change the line width of the plot to the system.”
8. “Estimate the effort for adding the ability to change the marker style of the plot to the system.”
9. “Estimate the effort for adding the ability to change the marker size of the plot to the system.”
10. “Estimate the effort for adding the ability to change the marker color of the plot to the system.”
11. “Estimate the effort for adding the ability to change the marker edge color of the plot to the system.”
12. “Estimate the effort for adding the ability to change the marker edge width of the plot to the system.”
13. “Estimate the effort for adding the ability to change the marker edge style of the plot to the system.”
14. “Estimate the effort for adding the ability to change the marker fill style of the plot to the system.”
15. “Estimate the effort for adding the ability to change the marker fill color of the plot to the system.”
16. “Estimate the effort for adding the ability to change the marker fill alpha of the plot to the system.”
17. “Estimate the effort for adding the ability to change the marker fill edge color of the plot to the system.”

18. “Estimate the effort for adding the ability to change the marker fill edge width of the plot to the system.”
19. “Estimate the effort for adding the ability to change the marker fill edge style of the plot to the system.”
20. “Estimate the effort for adding the ability to change the marker fill edge alpha of the plot to the system.”
21. “Estimate the effort for adding the ability to change the marker fill edge join style of the plot to the system.”
22. “Estimate the effort for adding the ability to change the marker fill edge cap style of the plot to the system.”
23. “Estimate the effort for adding the ability to change the marker fill edge mitre limit of the plot to the system.”
24. “Estimate the effort for adding the ability to include love numbers in the gravity-related calculations to the system.”
25. “Estimate the effort for adding the ability to include the effect of the atmosphere in the gravity-related calculations to the system.”
26. “Estimate the effort for changing the way the gravity-related calculations are done in the system.”
27. “Estimate the effort for changing the Doppler data to include the effect of the atmosphere on the system.”
28. “Estimate the effort for changing the Doppler data to include the effect of the atmosphere and the Love number on the system.”
29. “Estimate the effort for changing the way the Doppler data is processed in the system.”
30. “Estimate the effort for changing the way the Doppler data is processed to include the love number in the system.”

References

[1] Barry W. Boehm. *Software engineering economics*. Prentice-Hall advances in computing science and technology series. Englewood Cliffs, New Jersey: Prentice-Hall, 1981. ISBN: 978-0-13-822122-5.

[2] *COCOMO Model - Software Engineering*. GeeksforGeeks. Section: Software Engineering. Apr. 13, 2018. URL: https://www.geeksforgeeks.org/software-engineering-cocomo-model/ (visited on 03/07/2024).

[3] Aldo Dagnino. "Estimating software-intensive projects in the absence of historical data". In: *Proceedings of the 2013 International Conference on Software Engineering*. ICSE '13. San Francisco, CA, USA: IEEE Press, May 18, 2013, pp. 941–950. ISBN: 978-1-4673-3076-3. (Visited on 03/08/2024).

[4] Torleif Halkjelsvik and Magne Jørgensen. "From origami to software development: A review of studies on judgment-based predictions of performance time." In: *Psychological Bulletin* 138.2 (2012), pp. 238–271. ISSN: 1939-1455, 0033-2909. DOI: 10.1037/a0025996. URL: https://doi.apa.org/doi/10.1037/a0025996 (visited on 03/03/2024).

[5] M. Jørgensen. "A review of studies on expert estimation of software development effort". In: *Journal of Systems and Software* 70.1 (Feb. 1, 2004), pp. 37–60. ISSN: 0164-1212. DOI: 10.1016/S0164-1212(02)00156-5. URL: https://www.sciencedirect.com/science/article/pii/S0164121202001565 (visited on 03/07/2024).

[6] Luigi Lavazza et al. "Estimating Software Functional Size via Machine Learning". In: *ACM Transactions on Software Engineering and Methodology* 32.5 (July 21, 2023), 114:1–114:27. ISSN: 1049-331X. DOI: 10.1145/3582575. URL: https://dl.acm.org/doi/10.1145/3582575 (visited on 03/09/2024).

[7] Ayesha Saeed et al. “Survey of Software Development Effort Estimation Techniques”. In: *Proceedings of the 2018 7th International Conference on Software and Computer Applications*. ICSCA ’18. New York, NY, USA: Association for Computing Machinery, Feb. 8, 2018, pp. 82–86. ISBN: 978-1-4503-5414-1. DOI: 10.1145/3185089.3185140. URL: https://dl.acm.org/doi/10.1145/3185089.3185140 (visited on 03/08/2024).

[8] Martin Shepperd, Carolyn Mair, and Magne Jørgensen. “An experimental evaluation of a de-biasing intervention for professional software developers”. In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. SAC ’18. New York, NY, USA: Association for Computing Machinery, Apr. 9, 2018, pp. 1510–1517. ISBN: 978-1-4503-5191-1. DOI: 10.1145/3167132.3167293. URL: https://dl.acm.org/doi/10.1145/3167132.3167293 (visited on 03/08/2024).