Streamlining Software Ticket Handling: A Case Study with Transformers and Generative Models

Mohammad Bin Yousuf
School of Computer Science
CU 101239019 - Carleton University
Ottawa, Canada
mohammadyousuf@cmail.carleton.ca

Surendar Pala Danasekaran
Faculty of Engineering
uO 300401916 - University of Ottawa
Ottawa, Canada
spala021@uottawa.ca

Vrishab Prasanth Davey
Faculty of Engineering
uO 300438343 - University of Ottawa
Ottawa, Canada
vdave048@uottawa.ca

***Abstract*—This study presents an automated ticket classification, prioritization, and assignment system leveraging advanced Natural Language Processing (NLP) techniques and Large Language Models (LLMs). The system addresses the inefficiencies and errors inherent in manual triaging processes within software engineering project management. Employing transformer-based models such as fine-tuned BERT and generative AI models like GPT-4o, the system categorizes tickets, assesses their urgency, and assigns them to the appropriate teams. Comparative analysis evaluates the performance of these models across classification, prioritization, and team assignment tasks, revealing that fine-tuned BERT consistently outperforms GPT-4o, particularly in handling complex, multi-class problems. However, GPT-4o demonstrated potential with role-based prompting strategies for classification tasks. The study also explores challenges including class imbalance, textual complexity, and variability in LLM task handling. Insights gained suggest the potential for a hybrid approach combining the precision of fine-tuned transformers with the flexibility of generative AI, offering a scalable solution for efficient ticket triaging in dynamic project environments.**

***Index Terms*—natural language processing, large language models, classifiers, course project**

I. Introduction

This project aims to develop an automated ticket classification, prioritization, and assignment system utilizing advanced natural language processing (NLP) and large language models (LLM). Software Engineering projects frequently encounter a high volume of bug reports, suggestions, and change requests that vary significantly in type, severity, and clarity. These are usually managed with ticketing software like Jira, ZenDesk, or ServiceNow and are generally known as tickets. Manually triaging these reports is not only time-consuming but also prone to human error, often resulting in delayed responses to critical issues, especially in fast-paced development environments.

To address these challenges, this project employs two distinct approaches: one utilizing transformer-based models, such as fine-tuned BERT and Sentence Transformers, and the other leveraging generative AI, specifically GPT-4o. Unlike traditional classifiers, this project exclusively focuses on these advanced models to explore their effectiveness in automating the triaging process. The system will function in three key components.

In the first component, the system will classify tickets into predefined categories, either bugs or suggestions, based on a deep contextual understanding of the reports. In the second component, the model will prioritize the ticket by analyzing user-provided context to assess severity and impact, assigning highest (considered as critical), high, medium, or low priority levels. The third component will focus on automatically assigning the tickets to relevant development teams based on the description of the problem, ensuring that issues are efficiently directed to the appropriate resources.

A key focus of this project is to compare the performance of fine-tuned transformer-based models like BERT and Sentence Transformers against GPT-4o, evaluating their capabilities in classification, prioritization, and team assignment. Additionally, the project investigates whether these models agree or disagree in team assignments, offering insights into their practical applicability in real-world scenarios.

While the project initially aimed to include comparative evaluations with Claude and Gemini, subscription limitations prevented their inclusion. Despite this, the analysis will provide valuable insights into the performance differences between fine-tuned transformers and generative AI models.

The goal is to present a robust, industry-relevant solution for automating ticket triaging, addressing key challenges in software engineering project management. By streamlining classification, prioritization, and assignment processes, the system aims to enhance efficiency, reduce manual effort, and accelerate resolution timelines which ultimately would contribute to improved software quality.

In Section 2, we discuss the background of the problem and the past work related to the development of similar solutions. In Section 3, we discuss the technical approach and the various challenges encountered during the development of our system. In Section 4, we discuss the implementation steps and the technicalities of the designed system.

II. Background & Related Work

To develop a useful system, we must explore and understand what has been achieved in this domain. This literature review examines recent advancements in the application of Natural

Prof Mehrdad Sabetzadeh.
Fall 2024

Language Processing (NLP) and Large Language Models (LLMs) for automating tasks related to ticket management, such as classification, prioritization, and assignment.

Previous studies have explored the use of machine learning for ticket classification. Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Long Short-Term Memory (LSTM) networks have all been used to explore ticket classification due to their ability to remember dependencies in sequences of words [1]–[3]. However, there has been a shift towards transformer-based models. For instance, Ticket-BERT, a BERT model, demonstrates state-of-the-art performance in labelling incident management tickets, using a dataset that incorporates ticket titles, summaries, and multiple updates of ticket descriptions [4].

Ticket prioritization is a critical task in software development. Traditional methods of prioritization are often manual and are therefore time-consuming and prone to human error [1], [5]. Machine learning techniques can be used to automate this process, by utilizing Natural Language Processing (NLP) on the textual descriptions of tickets. For example, CNN-based approaches have been used to predict the multi-class priority of bug reports by preprocessing the textual information of the reports and conducting emotion analysis to compute an emotion value for each bug report [1]. Some research has also explored emotion-based methods for bug report prioritization [7]. Models for this task use a variety of methods for feature extraction including using word embeddings to represent the words in a ticket description [6] and then classifying the urgency of the bug using supervised learning algorithms [7]. Some researchers have used deep learning approaches for this task, finding these methods more effective than traditional approaches [8].

The automatic assignment of tickets to the appropriate team or team member is a well-explored area of research. Large Language Models (LLMs) such as GPT-4 are being explored for their ability to understand the problem described in tickets and assign them to the correct employee [9], [10]. This can be done using zero-shot, few-shot, or ensemble learning approaches, which are compared against a fine-tuned BERT model in one study [10]. Some approaches make use of Transformers, especially fine-tuned BERT models for ticket assignment, as well as classifiers to assign a ticket to one of many groups, and then to identify an expert within that group to solve the issue [11]. Some systems, like DeepRouting, use both a convolutional neural network for text matching and a graph convolutional network to determine the right team member to resolve a ticket [12]. Additionally, Sentence Transformer models can encode tickets into an embedding space to identify previously resolved, similar tickets for team members to inspect [11].

A key theme in recent research is the comparison of traditional transformer-based models with advanced LLMs. Studies show that fine-tuned LLMs, such as GPT-4o, can significantly outperform BERT in tasks such as sentiment analysis and star rating prediction of customer reviews, achieving an accuracy of 67% compared to BERT's 60.6% [13]. Another study shows that the GPT-4 model using in-context learning outperformed a fine-tuned GPT-3 model by an average of 24.7% in root cause analysis of cloud incidents [14]. The in-context learning approach leverages examples in the prompts to perform the root cause analysis rather than retraining or fine-tuning separately. This approach also resulted in notable improvements of 43.5% in correctness and 8.7% in readability in human studies conducted with incident owners [14]. While some studies have found that more traditional machine learning methods like logistic regression perform well in some use cases [15]–[18], it seems that the more recent transformer models like BERT and GPT have higher overall performance [4], [10], [13].

Despite the significant progress made over the last few years, there are several limitations and challenges. One issue is the need for large, high-quality training datasets, which may be difficult to acquire, especially for specific domains or languages [4], [10]. Data imbalances, where some categories are more frequent than others, can also affect model performance [15], [21]–[24]. Additionally, the reporting styles of different users can vary, and impact the flexibility of NLP models that are trained on particular data sets [15]. Another challenge is the dynamic nature of ticket data, where frequent revisions and updates necessitate models capable of adapting to these changes [4]. Also, the performance of LLMs can vary depending on the specific task being requested and the prompts being used, necessitating careful prompt engineering to elicit the best results [10], [19], [20].

These studies provide a solid foundation to further advance the use of LLMs in automating the classification, prioritization, and assignment of tickets. By leveraging LLMs, we aim to enable a more contextual and nuanced understanding of tickets that can improve the efficiency and accuracy of triaging processes.

III. Technical Approach

This section details the methodology adopted for developing a comprehensive ticket management system comprising three components: ticket classification, ticket prioritization, and team assignment. These components use advanced Natural Language Processing (NLP) and machine learning models, ensuring the system addresses the challenges of class imbalance, textual complexity, and scalability.

The core methodology employs transformer-based models for ticket classification and prioritization tasks, while a semantic similarity technique is applied for the team assignment task. For comparison, we evaluate the performance of a fine-tuned BERT transformer against GPT-4o across all three tasks.

A. Bug Classification

The ticket classification component categorizes tickets into predefined categories (bug or suggestion). The BERT-based approach employs a fine-tuned version of the ***bert-base-uncased*** model trained on the ticket dataset. Preprocessing involves tokenization, truncation, padding, and label encoding using the BERT tokenizer. The labels are mapped to numerical IDs to ensure compatibility with the model. The dataset is

divided into training and test sets, with the tokenized data converted into torch-compatible datasets.

The fine-tuning process optimizes the BERT model using a Trainer configured with hyperparameters, including a learning rate scheduler, weight decay, and warm-up steps. Performance is evaluated on the test set using precision, recall, F1 score, and accuracy.

For GPT-4o, the task involves prompt engineering by performing feature extraction relevant to the classification task. A training set guides the design of prompts, enabling GPT-4o to identify and extract key features from the ticket data for categorization. The test set is used to evaluate the model separately using the designed prompts. The comparative analysis evaluates the classification outcomes between the two models.

B. Bug Prioritization

The prioritization component predicts ticket urgency levels (highest, high, medium, low). The BERT-based approach utilizes the fine-tuned ***bert-base-uncased*** model trained on labeled ticket data, incorporating class weights to mitigate the impact of class imbalance. The preprocessing pipeline, training process, and evaluation method are the same as that used in classification.

For GPT-4o, prompt engineering is again employed to extract features relevant to prioritization. Prompts are iteratively refined based on a training set to guide GPT-4o in assigning priority levels. The results are compared against the BERT-based approach to assess performance differences.

C. Team Assignment

The team assignment component uses semantic similarity techniques to map tickets to appropriate development teams. Sentence Transformers encode both ticket descriptions and predefined team descriptions. The cosine similarity scores between ticket embeddings and team embeddings determine the best match.

Since ground truth labels for team assignment are unavailable, the BERT and GPT-4o predicted labels are compared for consistency. For GPT-4o, prompt engineering is tailored to facilitate a semantic understanding of ticket descriptions and team structures, enabling dynamic assignment predictions.

During the development process, several challenges were encountered. A significant issue was class imbalance, where the unequal distribution of samples across classes posed difficulties in the prioritization and classification tasks. Additionally, the complexity of textual data presented another challenge, as ticket descriptions often included domain-specific terminology and ambiguous language, making it difficult to extract and interpret meaningful insights.

These challenges are tackled during development. Class imbalance is tackled through class weights in the BERT-based models, and textual complexity is mitigated through advanced feature extraction techniques in both BERT and GPT-4o implementations. Scalability in team assignments is ensured by using a semantic similarity-based approach that adapts to dynamic team structures, avoiding the rigidity of traditional keyword-matching methods.

IV. Implementation

This section elaborates on the implementation of the ticket management pipeline, covering data preprocessing, model training, and the integration of its components into a cohesive and functional system. By adhering to Agile principles, the development process allowed concurrent progress across all components, enabling insights and features from one phase to inform and enhance the others.

A. Data Preprocessing

As with any NLP and ML initiative, data cleaning remains the most important step of the data preprocessing pipeline. To ensure consistency and quality across textual data, we developed and implemented a robust preprocessing pipeline.

Within the data preprocessing pipeline, all text fields from the Jira tickets were lower-cased for uniformity then all non-alphabetic content and special characters were removed to minimize noise in the dataset. Using NLTK's library, all stopwords were removed and the remaining words were tokenized to standardize the textual data. Since the Jira tickets contained both a Summary and Description field, we combined the fields into a unified feature for all further downstream tasks while retaining all contextual information. The preprocessed data served as a reliable and structured input for both BERT-based fine-tuning and GPT-4o prompt engineering.

B. BERT-based Approach

1) Ticket Classification: The Ticket Classification is the first component of the system. It utilizes a fine-tuned BERT transformer to categorize tickets into predefined classes. The tickets are either classified as a Bug or a Suggestion. To prepare the data for the model, the description data within the tickets were transformed into a format compatible with the ***bert-base-uncased*** model through tokenization, truncation, padding, and label encoding.

The fine-tuning process involved splitting the dataset into training and test sets and optimizing the model using a Trainer. Metrics such as precision, recall, accuracy, and F1 score were computed to evaluate the classifier's performance, providing a comprehensive view of its efficacy. The trained model and tokenizer were serialized for deployment, ensuring ease of integration with the larger system.

2) Ticket Prioritization: In the second component of the system, Ticket Prioritization, similar to Ticket Classification component, BERT was fine-tuned on preprocessed ticket descriptions. Custom class-weighted cross-entropy loss was utilized to address the imbalanced priority levels. The process involved tokenizing data with BERT's tokenizer, splitting the dataset into training, validation, and test sets (70%, 15%, 15%), and training for five epochs with early stopping to prevent over-fitting. Evaluation was performed using metrics such as accuracy, precision, recall, and F1-score, and confusion matrices were generated for error analysis. The fine-tuned model and tokenizer were, as before, saved for deployment.

3) Team Assignment: The final component of the system is the Team Assignment tool. It leverages a Sentence Transformer, ***all-MiniLM-L6-v2***, to compute semantic embeddings. Bug descriptions and team profiles were embedded, and cosine similarity was used to dynamically assign bugs to the most relevant teams. This approach is scalable, allowing seamless integration of new teams or changes in responsibilities.

To ensure that our tool is dynamic, users are allowed to define the functionality of the teams within their organization. For the purpose of demonstration, we defined the following teams:

- **UI (User Interface):** Handles frontend, user interface design, interaction problems, CSS, HTML issues, and component rendering errors.
- **Backend**: Responsible for server-side logic, API endpoints, database operations, backend crashes, latency, and authentication failures.
- **DevOps**: Focuses on infrastructure setup, CI/CD pipelines, cloud deployments, system configuration, containerization issues, and monitoring failures.
- **QA (Quality Assurance)**: Manages software testing, test automation, bug reporting, regression testing, performance issues, and quality assurance processes.

C. GPT-4o Approach

The implementation of the GPT-4o-based approach within the ticket management pipeline, focuses on the application of advanced prompt engineering techniques. The process involved splitting preprocessed data into training and testing sets, developing and refining prompts through iterative experimentation, and applying selected prompting strategies to achieve robust task performance.

1) Initial Prompt: The preprocessed dataset was divided into training and testing subsets in an 80:20 ratio. The initial prompt development was guided by manual inspection and analysis of the dataset, allowing the team to gain an understanding of the ticket content and tailor it to the task objectives. Through iterative discussions, the team determined the desired outputs and identified critical aspects for the model to focus on during classification, prioritization, and assignment tasks.

2) Prompting Strategies: Four prompting strategies were selected from a broader set of eight researched techniques based on their relevance and potential to address the complexities of the tasks. These strategies included Zero-Shot Prompting, Chain-of-Thought Prompting, Tree-of-Thought Prompting, and Role-Based Prompting, each offering unique strengths to guide the GPT-4o model effectively. For instance, Zero-Shot Prompting leveraged the model's inherent generalization capabilities without requiring specific task examples, while Chain-of-Thought and Tree-of-Thought approaches promoted logical reasoning and deliberate problem-solving through structured intermediate steps. Role-based prompting enhanced task alignment by framing the model's responses through a defined lens, such as acting as a domain expert.

For each selected strategy, an engineered prompt was developed and tested independently to ensure isolated evaluation. This approach avoided potential cross-contamination across strategies, maintaining the integrity of the results. The process involved applying the engineered prompts to the test dataset, allowing the model to generate predictions for each ticket. The prompts were iteratively refined to align with task-specific requirements.

Chain-of-Thought prompting [25] encouraged the model to break down complex problems into sequential logical steps, thereby enhancing transparency and reasoning. Similarly, Tree-of-Thought Prompting [26] facilitated systematic exploration of problem-solving pathways through intermediate language sequences, employing search algorithms to optimize decision-making. Zero-Shot Prompting [27] demonstrated the model's capacity to handle tasks without explicit examples, relying instead on its general understanding of language. Finally, Role-Based Prompting [28] framed the model's responses through a specific role, shaping outputs to align with domain-specific expectations.

While the focus remained on the four selected strategies, other prompting techniques were also explored in the preliminary stages of the project. Meta-Prompting [29], Few-Shot Prompting [30], Generated Knowledge [31], and Self-Consistency [32] were reviewed for their usefulness. These techniques, though valuable, were not implemented due to scope constraints, time constraints, and their lack of application to the specific requirements of the pipeline.

Each engineered prompt underwent rigorous testing within the GPT-4o framework to evaluate its performance on the test dataset. Metrics such as precision, recall, accuracy, and F1-score were employed to assess task-specific outcomes, ensuring alignment with the overarching goals of classification, prioritization, and team assignment. The results from the selected strategies were analyzed to determine if it was possible to integrate into the dynamic ticket management system.

V. Empirical Evaluation

In this section, we focus on assessing the performance of the ticket management pipeline across its core components, using our dataset and the aforementioned evaluation metrics.

A. Research Questions

The research questions were refined during the project's planning phase to align with its objectives and to evaluate the efficacy of leveraging large language models (LLMs) and machine learning models for automated ticket management tasks. These questions also consider the robustness, ambiguities, and biases inherent in LLMs. During our development cycle, the Research Questions (RQs) were updated to match the scope of the project. The project were guided by the following questions:

- **RQ1:** How effectively can Large Language Models (LLMs) classify user-reported ticket into predefined categories?

- **RQ2:** How accurately can Large Language Models (LLMs) prioritize user-reported ticket by severity, addressing issues of class imbalance and contextual nuances?
- **RQ3:** How can the limitations of Large Language Models (LLMs), including biases and interpretability challenges, be mitigated or managed in the context of ticket management?

B. Data Set

The dataset was extracted from the Jira issue tracking system for the ***Sourcetree for Windows*** project, a tool designed to simplify interactions with Git repositories through a visual interface.

The dataset mined contains approximately 10,090 issues related to software development that range from the year 2012 till October 2024 therefore fetching the latest dataset, categorized into various types such as bugs and and suggestions. Using the Jira REST API, the issues were retrieved in batches of 1000 due to the API's limitations and then exported the data into a CSV file for further analysis and cleaning of the data as most of the fetched fields were irrelevant.

This dataset includes key fields such as Issue Key, Issue Type, Summary, Status (e.g., Open, Closed, Resolved), Description, Creation Date, Resolution Date, and Assignee details among others. The analysis will focus on basic data-cleaning tasks, such as handling missing or irrelevant fields such as nested objects within other objects, standardizing values, and removing outliers. The dataset offers deep insights into the life cycle of software development tasks, helping to track how bugs or suggestions are identified, reported, and resolved over time. By analyzing this real-world dataset, the aim is to explore issue resolution trends, task prioritization, and team assignment in a software project setting, focusing on improving project management and bug-triaging practices.

The data provides a rich foundation for applying these Large Language Models (LLMs) to classify and prioritize issues automatically, hopefully enabling better project management in software development.

C. Analysis Procedure and Metrics

The project uses a systematic approach to assess the effectiveness of the ticket management pipeline across classification, prioritization, and assignment tasks. For classification and prioritization, fine-tuned BERT models and GPT-4o predictions are compared against ground truth labels using metrics such as accuracy, precision, recall, and F1-score. The semantic similarity-based assignment approach, utilizing Sentence Transformers, evaluates team assignment consistency, relying on cosine similarity between ticket and team embeddings. For GPT-4o, predictions are generated through structured prompt engineering, and model outputs are compared with BERT predictions to analyze discrepancies and potential biases.

As mentioned before, to ensure robustness, the analysis incorporates methods to address challenges such as class imbalance and textual complexity. Class weights mitigate imbalance

```
1. Analyze the following dataset (jira_train.csv), understand how
   the "priority" is assigned to the "description". This is done
   for the "Bug prioritization task", similarly understand how the
   "issue_type" is assigned to the particular description. For example:
   - "Highest" : "crash push sourcetree crash often pushpull action orign"
   - "Low" : "show explorer reopens existing folder clicking explorer
     button selecting show explorer menu opens folder working copy folder
     already open pressing button opens another instance folder instead
     bringing folder"
   - "Medium" : "sourcetree frequently hangs performance issues team
     developers use sourcetree large repository tens thousands files
     often branch merge first started use soucetree around v everybody
     thrilled satisfied however since diff viewer new file status view
     introduced v us experience unpleasant lags multiple seconds frequent
     hangs kill application although new views look much nicer seem
     update often use considerably time memory many us frustrated point
     look alternative clients revert old versions appreciate sourcetree
     work guys put wed love continue using please consider improve
     performance versions add features related issue srctreewin"
   "The keywords are necessary to prioritize," for instance:
   - "crashes," "login," or "password" may indicate High priority.
   - Terms like "files" or "update" often suggest Medium or Low priority.
   Similarly for classification the relation is as follows:
   - "Suggestion" : "add quick search filter top projects pane retrieve
     many projects"
   - "Bug": "add unstaged files time using checkbox top"
   "The keywords are necessary to classify," for instance:
   - For issue type classification:
     - Keywords like "add," "useful," "great," or "easier" suggest a
       Suggestion.
     - Keywords like "crashes," "working," or "open" suggest a Bug.
```

Fig. 1. Zero-Shot Prompting

in BERT models, while iterative prompt refinement enhances GPT-4o's understanding of domain-specific terminology.

1) Zero-Shot Prompting: Zero-shot prompting involves instructing the model to perform a task without providing examples, relying solely on the task's description. In our implementation (Figure 1), we explicitly described the relationship between keywords in bug descriptions and their respective classifications or priorities. The model used these descriptions to infer the labels directly, demonstrating its ability to generalize from context without explicit examples.

2) Chain-of-Thought Prompting: Chain-of-Thought (CoT) prompting guides the model to reason step-by-step, breaking down complex tasks into smaller, logical steps. In our design (Figure 2), the prompts directed the model to first analyze keywords, then assign classifications or priorities, and finally recommend team assignments. This structured reasoning approach ensured that the model's outputs were interpretable and aligned with human decision-making processes.

3) Tree-of-Thought: Tree-of-Thought (ToT) prompting simulates a collaborative reasoning process, where multiple "analysts" evaluate, refine, and reach a consensus. In our implementation (Figure 3), we instructed the model to simulate independent evaluations by virtual analysts for classification, prioritization, and team assignment. This approach allowed the model to explore diverse reasoning paths and converge on the most robust solutions.

4) Role-based Prompting: Role-based prompting assigns the model a specific personality, leveraging domain expertise to enhance task performance. In our implementation (Figure 4), the model was instructed to act as a Lead Software Engineer, using its "expertise" to analyze descriptions, assign

TABLE I
PROMPT STRATEGY METRICS (MARCO AVERAGE)

Prompt Strategy	Task	Precision (%)	Recall (%)	F1-Score (%)	Overall Accuracy (%)
Zero-Shot	Classification	68.0	61.0	62.0	77.0
	Prioritization	30.0	30.0	27.0	40.0
Chain of Thoughts	Classification	59.0	60.0	46.0	46.0
	Prioritization	25.0	28.0	26.0	39.0
Tree of Thought	Classification	68.0	65.0	66.0	77.0
	Prioritization	30.0	26.0	19.0	22.0
Role Based Prompting	Classification	85.0	78.0	81.0	87.0
	Prioritization	42.0	28.0	25.0	54.0

```
Analyze the following dataset (jira_train.csv), understand how
the "priority" is assigned to the "description". Perform the Classification
task on (jira_test_Classification.csv) and the Prioritization task on
(jira_test_priority.csv) step-by-step, focusing only on the “description” column:
1. Classification Task:
   - First, identify the context and keywords in each ticket description.
   - Then, determine whether the description suggests a “bug” or a
     “suggestion” based on the provided keyword guidelines.
   - Finally, assign the appropriate label (“bug” or “suggestion”) in a new
     column called predicted_issue.
2. Prioritization Task:
   - First, analyze the urgency and criticality of the description content.
   - Then, assign a priority level (“highest,” “high,” “medium,” or “low”)
     based on the keyword guidelines.
   - Finally, save the assigned priority level in a new column called
     predicted_priority.
3. Team Assignment Task:
   - Identify the technical domain or responsibility area implied by the
     description.
   - Match it to one of the following teams based on relevance: UI, Backend,
     DevOps, QA.
   - Assign the ticket to the most appropriate team in a new column called
     predicted_teamassignment.
```

Fig. 2. CoT Prompting

```
Imagine three virtual analysts are collaborating to analyze the
(jira_train.csv) dataset and focus on the “description” column. The analysts
will work step-by-step, evaluating and refining their thoughts at each stage.
If an analyst realizes theyve made an incorrect decision, they will step back
and revise their approach.
1. Classification Task:
   - Each analyst will evaluate the ticket description and suggest whether it
     is a “bug” or a “suggestion” based on contextual clues and provided
     keyword guidelines.
   - The final decision for predicted_issue will be based on consensus or
     majority reasoning.
2. Prioritization Task:
   - Each analyst will independently assign a priority level (“highest,”
     “high,” “medium,” or “low”) based on the description's urgency.
   - The final decision for predicted_priority will reflect the most robust
     reasoning.
3. Team Assignment Task:
   - Each analyst will independently match the ticket description to a team
     (“UI,” “Backend,” “DevOps,” or “QA”) by evaluating the technical domain.
   - The final decision for predicted_teamassignment will represent the
     strongest reasoning path.
```

Fig. 3. Tree-of-Thought Prompt

classifications and priorities, and make team assignments. This strategy grounded the model's reasoning in a professional context, enhancing the relevance of its outputs.

Results are analyzed not only for task-specific performance but also for their broader implications in automating ticket triaging, focusing on practical applicability and alignment with industry needs. Accuracy, precision, recall, and F1-score are

```
You are a Lead Software Engineer specializing in issue tracking
and team collaboration. Using your expertise, analyze the (jira_train.csv)
dataset and understand how the "priority" is assigned to the "description".
After this, perform the “Classification task” on (jira_test_Classification.csv)
and the “Prioritization task” on (jira_test_priority.csv).
Tasks:
1. Classification Task:
   - Determine whether each ticket is a “bug” or a “suggestion” based on the
     description content. Record your decision in a new column called
     predicted_issue.
2. Prioritization Task:
   - Assign a priority level (“highest,” “high,” “medium,” or “low”) to each
     ticket based on its urgency and potential impact. Record your assignment
     in a new column called predicted_priority.
3. Team Assignment Task:
   - Assign each ticket to the most relevant team (“UI,” “Backend,” “DevOps,”
     or “QA”) based on the description content. Record your decision in a new
     column called predicted_teamassignment.
```

Fig. 4. Role-Based Prompting

key metrics that are used for evaluation. Accuracy measures the overall correctness while Precision focuses on the quality of positive predictions. Recall assesses the model's ability to identify all relevant instances and the F1-score provides a balanced measure when dealing with imbalanced datasets. These metrics offer a comprehensive evaluation of a model's performance.

D. Results & Discussion

TABLE II
BERT (MARCO AVERAGE) METRICS

Model	Precision (%)	Recall (%)	F1-Score (%)
BERT Classification	90.1	90.3	90.2
BERT Prioritization	54.7	55.7	55.0

1) Results: The results from our experimentation with a BERT-based vs GPT-based model for text classification and prioritization tasks highlight the strengths and limitations of each approach. These outcomes, summarized in Tables I and II, provide insights into the efficacy of prompt engineering compared to a fine-tuned BERT model.

To ensure consistency in evaluation and to provide a balanced perspective, we rely on macro averages for our reported metrics. Macro averages calculate the unweighted mean of performance scores across all classes, treating each class equally, regardless of its frequency in the dataset. This evaluation

method addresses the issue of class imbalance inherent in our dataset, which had underrepresented classes. By focusing on macro averages, we emphasize the models' ability to handle all classes equally, rather than being influenced by the performance of more frequent classes.

The Zero-Shot prompting strategy (Figure 1) showed moderate success in classification tasks, achieving a macro average F1-score of 62% and an overall accuracy of 77%. However, its performance on prioritization tasks was considerably lower, with an F1-score of 27% and an accuracy of 40%. This demonstrates the limitations of zero-shot methods when addressing tasks that require a deeper contextual understanding.

Chain of Thoughts (CoT) (Figure 2) prompting exhibited similar trends, with a classification F1-score of 46% and accuracy of 46%, while its prioritization results were comparable to Zero-Shot, achieving an F1-score of 26%. While Chain-of-Thought encourages step-by-step reasoning, its effectiveness appears constrained when complex patterns are involved.

Tree-of-Thought (ToT) (Figure 3) prompting yielded better results for classification, with an F1-score of 66% and accuracy of 77%. However, its prioritization performance was the weakest among all strategies, with an F1-score of 19% and an accuracy of 22%. The collaborative reasoning approach of Tree-of-Thought benefited the classification task but struggled with multi-faceted tasks like prioritization.

Role-based (Figure 4) prompting emerged as the most effective among the prompt-based strategies. It achieved an F1-score of 81% and an accuracy of 87% for the classification task, significantly outperforming other methods. While its prioritization results (F1-score of 25%) were still modest, the expert framing of this strategy seems to enhance performance by grounding the model in a domain-specific context.

The fine-tuned BERT model surpassed all prompting strategies across both tasks. For classification, BERT achieved an F1-score of 90.2% and recall of 90.3%, indicating its ability to robustly identify patterns in the dataset. On the prioritization task, BERT's F1-score of 55.0% was significantly higher than the best-performing prompting strategy (Role-Based Prompting at 25%). These results underscore the power of fine-tuned language models in capturing domain-specific nuances.

To assess the consistency of the different prompting strategies for team assignment, we measured inter-strategy agreement using Cohen's Kappa. This metric evaluates the extent to which two strategies align beyond chance. The findings, presented in Table III, reveal patterns of alignment and disagreement that provide insight into the reliability and compatibility of these strategies.

The findings indicate minimal alignment across strategies, with the average Kappa score being 0.067, reflecting very low agreement overall. Among the pairwise comparisons, the highest agreement was observed between the Chain of Thought and Zero-Shot strategies (Kappa = 0.375), categorized as fair agreement. Slight agreement was noted between the Chain of Thought and Tree of Thought strategies (Kappa = 0.137).

The remaining comparisons demonstrated minimal or no agreement. For instance, agreements involving Role-Based Prompting were particularly low, with Cohen's Kappa values close to zero for most pairs. Similarly, agreements with the Sentence Transformer strategy, such as with Zero-Shot and Tree of Thought, also yielded Kappa values close to zero. These results highlight a lack of consistency among the various strategies evaluated.

2) Discussion: While prompting strategies offer an easier and simpler alternative to fine-tuning, their performance depends heavily on the complexity of the task. Role-based prompting's superior classification results suggest that task framing and context play a critical role in maximizing model performance. However, the consistently low prioritization scores across all prompting strategies highlight their limitations in handling intricate multi-class problems.

Overall, the GPT models had a difficult time understanding the priority in this use case. We used a cleaned dataset where stop words were removed to ensure consistency and a fair comparison across models. While this approach allowed us to evaluate both systems under identical conditions, it is worth noting that GPT models, being generative, are often more attuned to processing complete English sentences. The removal of stop words may have inadvertently affected its performance, as its pre-training and architecture are optimized for natural language patterns, including grammatical constructs and connective words. This is to ensure that the results are reflective of the models' inherent capabilities rather than preprocessing advantages or disadvantages.

However, the fine-tuned BERT model demonstrated its dominance in both classification and prioritization tasks. This aligns with expectations, given its ability to learn task-specific representations during training. At the end of the day, the technical knowledge, computational cost, and resource requirements of fine-tuning make prompting strategies an appealing option for scenarios where resources are constrained.

The results in Table III reveal critical insights into the inter-strategy agreement of different prompting approaches. The relatively higher agreement between Chain of Thought and Zero-Shot strategies (Kappa = 0.375) suggests some level of methodological alignment, potentially due to their approach of sequential reasoning or generalizable task-specific outputs. Conversely, the low agreement scores for Role-Based Prompting and Sentence Transformer strategies indicate that these approaches generate outputs with distinct interpretations or representations compared to others.

The overall low average Kappa value (0.067) highlights the divergence in strategy outcomes, underscoring the challenges of consistency in NLP tasks when employing diverse prompting techniques. This low agreement could be attributed to differences in how each strategy handles nuances within the dataset or their respective sensitivity to task framing. For instance, strategies like Role-Based Prompting, designed for structured reasoning, may differentiate significantly when compared to simpler methods like Zero-Shot prompting.

TABLE III
PAIRWISE AGREEMENT BETWEEN PROMPTING STRATEGIES (COHEN'S KAPPA)

Strategy Pair	**Agreement Level (Kappa)**	**Interpretation**
Chain of Thought vs. Role-Based	-0.031	Slight Disagreement
Chain of Thought vs. Tree of Thought	0.137	Slight Agreement
Chain of Thought vs. Zero-Shot	0.375	Fair Agreement
Chain of Thought vs. Sentence Transformer	0.048	Almost No Agreement
Role-Based vs. Tree of Thought	0.0	No Agreement
Role-Based vs. Zero-Shot	0.024	Almost No Agreement
Role-Based vs. Sentence Transformer	0.019	Almost No Agreement
Tree of Thought vs. Zero-Shot	0.038	Almost No Agreement
Tree of Thought vs. Sentence Transformer	0.030	Almost No Agreement
Zero-Shot vs. Sentence Transformer	0.027	Almost No Agreement
Overall Average Agreement	**0.067**	**Very Low Agreement**

E. Threats to Validity

Several factors may have influenced the validity of the results in this study. First, the dataset used for training showed significant class imbalance, with certain categories being overrepresented while others were underrepresented. This imbalance posed challenges for model performance, potentially leading to better outcomes for more frequent classes and poorer results for less frequent ones. Although BERT-based models incorporated class weights to address this issue, the imbalance remained a potential threat to the robustness of the findings.

The complexity and quality of the textual data also introduced validity concerns. Ticket descriptions often contained domain-specific terminology, ambiguous phrasing, and inconsistent reporting styles, which made extracting meaningful insights challenging. Despite the implementation of a robust preprocessing pipeline, the real-world nature of ticket data introduced noise and inconsistencies that could affect model performance. Additionally, the reliance on prompt engineering in the GPT-4o-based approach further underscored the sensitivity of results to the design and quality of prompts. While role-based prompting yielded the best outcomes, the variability in performance across different strategies highlighted the limitations of this approach.

Finally, the generalization of the findings remains limited due to the dataset's specific context—Jira tickets from the SourceTree for Windows project. This dataset may not represent the characteristics of other projects or domains, restricting the broader applicability of the results. Moreover, the use of large language models (LLMs) introduced potential biases and interpretability challenges, as their "black-box" nature makes it difficult to understand decision-making processes. The consistently low prioritization scores across all prompting strategies further emphasized the constraints of prompt-based methods in handling complex multi-class problems, which may limit their scalability to similar tasks.

VI. LESSONS LEARNED

The development and evaluation of the automated ticket classification, prioritization, and assignment system provided valuable insights into leveraging advanced Natural Language Processing (NLP) and Large Language Models (LLMs).

Fine-tuned transformer models, such as BERT, demonstrated high efficacy in classification tasks, leveraging their ability to capture contextual information while struggling with prioritization in imbalanced datasets. GPT-4o, with its reliance on prompt engineering, showcased both potential and limitations, excelling in Role-Based Prompting for classification but underperforming in prioritization tasks requiring deeper contextual understanding.

The robust preprocessing pipeline and feature engineering significantly enhanced data quality, ensuring consistency and contextual richness, which were crucial for downstream performance in the BERT-based approach. Addressing class imbalance using weighted loss functions improved BERT's robustness, whereas prompt-based methods struggled.

Semantic similarity using Sentence Transformers proved effective and scalable for dynamic team assignments, demonstrating practical applicability. Comparing BERT and GPT-4o highlighted differences in biases, interpretability, and variability, which suggests that a hybrid approach combining the precision of BERT with the flexibility of GPT-4o may yield the best results.

VII. CONCLUSION

This study investigated the effectiveness of fine-tuned transformer models (BERT) and generative AI models (GPT-4o) for automating ticket classification, prioritization, and assignment. The findings demonstrated that fine-tuned BERT models consistently outperformed GPT-4o, particularly in addressing complex, multi-class problems, underscoring the advantages of task-specific training. While GPT-4o exhibited potential with role-based prompts for classification, its performance in prioritization tasks highlighted the limitations of prompt engineering for handling complex and intricate tasks.

The analysis also recognized several factors that could influence the validity of the results, including class imbalance, textual complexity, and dataset-specific constraints. These limitations provide important directions for future research. In particular, further exploration of prompt engineering techniques with more human-readable datasets could enhance the application of generative AI models in contexts requiring deep contextual understanding. In future work, a hybrid approach that integrates the precision of fine-tuned transformers with the adaptability of generative AI could improve the efficiency and scalability of automated ticket management systems.

A. Replication Package

The implementation details and code base of this project are publicly available in a dedicated GitHub repository. The replication package provides comprehensive documentation, including data preprocessing scripts and model training workflows. The replication package can be accessed at https://github.com/surendar-pd/CSI5137-Final-Project.

REFERENCES

[1] Q. Umer, H. Liu, and I. Illahi, "CNN-based automatic prioritization of bug reports," IEEE Trans. Rel., vol. 69, no. 4, pp. 1341–1354, Dec. 2020.

[2] B. Wang, "Disconnected recurrent neural networks for text categorization," in Proc. 56th Annu. Meeting Assoc. Comput. Linguistics (Volume 1: Long Papers), Melbourne, VIC, Australia, Jul. 2018, pp. 2311–2320.

[3] A. Revina, K. Búza, V. G. Meister, "IT ticket classification: The simpler, the better," IEEE Access, 8, 193380–193395, 2020.

[4] Z. Liu, C. Benge, and S. Jiang, "Ticket-BERT: Labeling Incident Management Tickets with Language Models," Unpublished.

[5] P. Choudhary, "Neural network based bug priority prediction model using text classification techniques," Int. J. Adv. Res. Comput. Sci., vol. 8, no. 5, pp. 1315–1319, 2017.

[6] K. H. Alyoubi and F. S. Alotaibi, "A novel multi-layer feature fusion-based BERT-CNN for sentence representation learning and classification," Robotic Intelligence and Automation, vol. 43, no. 6, pp. 704-715, 2023.

[7] Q. Umer, H. Liu, and Y. Sultan, "Emotion based automated priority prediction for bug reports," IEEE Access, vol. 6, pp. 35743–35752, 2018.

[8] M. Kumari and V. B. Singh, "An improved classifier based on entropy and deep learning for bug priority prediction," in Proc. 18th Int. Conf. Intell. Syst. Design Appl. (ISDA), 2018, pp. 571–580.

[9] P. Zicari, G. Folino, M. Guarascio, L. Pontieri, "Discovering accurate deep learning based predictive models for automatic customer support ticket classification," in Proceedings of the 36th Annual ACM Symposium on Applied Computing, SAC '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 1098–1101.

[10] N. Arici, L. Putelli, A. E. Gerevini, L. Sigalini and I. Serina, "LLM-based Approaches for Automatic Ticket Assignment: A Real-world Italian Application," Unpublished.

[11] Y. Feng, Y. Lu, R. Jiang, W. Liu, H. Wang and L. Chen, "TaDaa Real Time Ticket Assignment Deep Learning Auto Advisor for Customer Support, Help Desk, and Issue Ticketing Systems", Unpublished.

[12] J. Han, A. Sun, Deeprouting: A deep neural network approach for ticket routing in expert network, in: 2020 IEEE International Conference on Services Computing, SCC 2020, Beijing, China, November 7-11, 2020, IEEE, 2020, pp. 386–393.

[13] K. Roumeliotis, D. Nasopoulos and S. Dimna, "Leveraging Large Language Models in Tourism: A Comparative Study of the Latest GPT Omni Models and BERT NLP for Customer Review Classification and Sentiment Analysis", Unpublished.

[14] Zhang et al., "Leveraging GPT-4 Model on Root Cause Analysis through In-Context Learning", Unpublished.

[15] E. Yang, M. D. Li, S. Raghavan, F. Deng, M. Lang, M. D. Succi, et al, "Transformer versus traditional natural language processing: how much data is enough for automated radiology report classification?", Br J Radiol (2023).

[16] C. A. Chilakapati, "Bow vs BERT: classification", Data Exploration, 2019.

[17] Tan et al., "Comparison of natural language processing rules- based and machine- learning systems to identify lumbar spine imaging findings related to low back pain", Acad Radiol, 2018.

[18] J. T. Senders, A. V. Karhade, D. J. Cote, A. Mehrtash, N. Lamba, A. DiRisio, et al. Natural language processing for automated quantification of brain metastases reported in free- text radiology reports. JCO Clinical Cancer Informatics, 2019.

[19] T. Shin, Y. Razeghi, R. L. L. IV, E. Wallace, S. Singh, "Autoprompt: Eliciting knowledge from language models with automatically generated prompts," in: Proceedings of the 2020 Confference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020, Association for Computational Linguistics, 2020, pp. 4222–4235.

[20] M. A. Sami, Z. Rasheed, M. Waseem, Z. Zhang, T. Herda and P. Abrahamsson, "Prioritizing Software Requirements Using Large Language Models," Unpublished.

[21] N. Z. Bawany, A. Azhar, S. Q. Gilani, N. Abbas, and I. Lali, "CaPBug-A Framework for Automatic Bug Categorization and Prioritization Using NLP and Machine Learning Algorithms", Unpublished

[22] N. K. Singha Roy and B. Rossi, "Cost-sensitive strategies for data imbalance in bug severity classification: Experimental results," in Proc. 43rd Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA), Aug. 2017, pp. 426–429

[23] R. Shu, T. Xia, L. Williams, and T. Menzies, "Better security bug report classification via hyperparameter optimization," 2019, arXiv:1905.06872.

[24] T. Hirsch and B. Hofer, "Using textual bug reports to predict the fault category of software bugs," Array, 15, 100189, 2022

[25] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al., "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24824–24837, 2022. Available: https://arxiv.org/pdf/2201.11903

[26] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024. Available: https://arxiv.org/pdf/2305.10601

[27] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *arXiv.org*, May 24, 2022. Available: https://arxiv.org/abs/2205.11916

[28] A. Kong et al., "Better zero-shot reasoning with role-play prompting," *arXiv.org*, Aug. 15, 2023. Available: https://arxiv.org/abs/2308.07702

[29] M. Suzgun and A. T. Kalai, "Meta-Prompting: Enhancing language models with task-agnostic scaffolding," *arXiv.org*, Jan. 23, 2024. Available: https://arxiv.org/abs/2401.12954

[30] T. Brown et al., "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020. Available: http://arxiv.org/pdf/2005.14165

[31] J. Liu et al., "Generated knowledge prompting for commonsense reasoning," *arXiv.org*, 2021. Available: https://arxiv.org/pdf/2110.08387

[32] X. Wang et al., "Self-consistency improves chain of thought reasoning in language models," *arXiv.org*, 2022. Available: https://arxiv.org/pdf/2203.11171