ΟΙΚΟΝΟΜΙΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΘΗΝΩΝ

ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS

BUSINESS
ANALYTICS
Master of Science

# MSc Business Analytics

Summer Semester 2025-2026

**Course:** Machine Learning & Content Analytics

**Project Team Name**: Papa

**Project Title:** Agentic AI Ticket Processing System

**Students**:

Argyriou Christos (p2822402)

Gkinos Anastastios (p2822405)

Panagiotidis Diamantis (p2822421)

Sarris Ioannis (p2822430)

# 1. Introduction

Nowadays, customer service quality is a critical differentiator for businesses, but many organizations face significant internal inefficiencies due to manual ticket handling. This traditional approach creates significant delays and is prone to errors, failing to meet modern customer expectations for rapid response times. The goal of this project is to address these challenges by developing an intelligent, AI-powered system designed to automate the entire ticket triaging workflow. By implementing advanced natural language processing (NLP) models, this system accurately classifies incoming support tickets by department, type, and relevant tags, routing them to the appropriate team for optimal resolution.

Transitioning to an AI-powered triage system has significant benefits. By automating the classification process, organizations can significantly reduce operational costs, enhance human-agent productivity, and improve the overall customer experience, leading to measurable gains in customer satisfaction and retention. Instead of maintaining a complex and fragile set of rules, an AI model can be easily updated by retraining it on new data, ensuring the system remains aligned with evolving business needs and scales effectively with a growing volume of tickets.

The structure of this report is as follows: Section 2 identifies the business need and the benefits of this system for an organization. Section 3 analyzes the customer support ticket dataset, detailing the collection, preprocessing, and cleaning procedures. Section 4 presents the implementation and evaluation of four separate classification models for predicting ticket department, tags, and type, comparing baseline approaches against fine-tuned Transformer models. Section 5 discusses the implementation and results of an alternative multi-task learning model. Section 6 describes the development of the final agentic workflow, which orchestrates these models to create a fully autonomous ticket processing pipeline. Finally, the report finishes with Sections 7 and 8 in which the conclusion and appendix of the project are presented, respectively.

## 2. Business Need

The internal inefficiencies of manual triage translate into a degraded customer experience with measurable implications for retention and revenue. Fast response time is the strongest driver of perceived service quality and plays a crucial role in customers' views about the company and their engagement with it in the near future. Manual triage is not an acceptable way to handle customer tickets. It requires a significant amount of workforce to handle them 24/7, which increases the cost and also creates a big gap between customer expectations and actual performance. A persistent gap separates customer expectations from typical performance.

Nearly half of customers expect a response within **four** hours (SuperOffice). In practice, average response times among firms that do reply exceed **twelve hours**, and some extend beyond **eight days**; approximately **62%** do not reply to service emails at all (SuperOffice). This chasm is a direct result of queues that are manually managed, where tickets sit waiting to be read, assessed, and routed. This is also costly for the company, since it creates unsatisfied customers who are more likely to stop using the company's services or products, and also discourages others from selecting them, hurting the brand image in the process. This chasm is a direct result of queues that are manually managed, where tickets sit waiting to be read, assessed, and routed.

On the other hand, many companies have already transitioned to automations that rely on rule-based automation. These processes are a better solution to the problems described above, but still fail to meet customer needs. These systems are fundamentally fragile. They require exact keyword matches to function, meaning a rule designed to find keywords like "Can not sign in" to describe the inability of a user to connect to their account. In addition to synonyms, possible typos or ironies that could be misleading are additional challenges that these systems have to deal with.

The biggest problem with these systems, though, is that they are a significant maintenance burden. As the business adds new products, features, or policies, the list of rules grows exponentially. This quickly evolves into a tangled, convoluted mess of hundreds of rules that become difficult to manage, impossible to audit, possibly contradict each other, and too risky for anyone to modify. Each new customer and each new product adds a disproportionate amount of operational friction, risk, and cost, which prevents sustainable growth.

The transition to an AI-powered intelligent triaging classification mechanism confronts all the aforementioned challenges. This system has the ability to learn from historical ticket data and apply this knowledge in the upcoming ticket inquiries, while updating its system with any additional information provided. This addresses the problems of the old systems regarding operational speed since it enhances the customer experience, leading to tangible results such as a **15%** drop in negative sentiment, and customer satisfaction (CSAT) scores reaching as high as **96%** (ClientZen, Sobot.io). Moreover, the financial business case is equally strong, with organizations reporting a **25-45%** reduction in overall support operation costs since fewer human agents would be needed to solve the respective case and not spend time checking, reading, and evaluating aspects of the ticket (Medium).

As already mentioned, the traditional systems require significant maintenance and updating-related costs. Instead of redesigning workflows or rewriting business rules, an AI-driven system can be improved only by retraining the model with updated data. As a consequence, the firm is constantly aligned with the increased volume of tickets during its expansion. This is especially beneficial for fast-growing businesses that have an increasing flow of tickets and cases to manage. This way, it is ensured that the company will not collapse and will provide a high-quality customer service experience.

Ultimately, all these enhancements result in higher customer satisfaction and increased brand loyalty. Customers who do not get a response or do not get a sufficient solution to their problem are more likely to leave the company and turn to the competition. This AI-powered system comes to meet customer needs regarding fast responses, which is critical to customer retention and helps to create strong and long-term relationships.

## 3. Data

This section provides a structured overview of the dataset used in this project. It begins by describing the source and nature of the data, followed by the preprocessing and cleaning procedures applied to ensure its quality and suitability for analysis. The section then presents the final dataset schema and characteristics, complemented by visual explorations that highlight key distributions, trends, and patterns within the customer support tickets.

## 3.1 Data Collection

The dataset is derived from the “Multilingual Customer Support Tickets” collection, publicly available on the Kaggle data science platform. This collection was specifically designed to support the development and evaluation of machine learning models for help-desk automation tasks, such as ticket classification and routing. An important characteristic of this dataset is its synthetic nature, as the tickets were artificially generated rather than collected from real customer interactions.

## 3.2 Data Preprocessing and Cleaning

The original dataset consisted of **28,587** records with **16** columns. However, several transformations and cleaning steps were necessary before conducting analysis and modeling. As a first step, filtering was applied to retain only the English-language tickets, allowing for a more focused and consistent analysis.

Moreover, several text-cleaning transformations were applied to the dataset to ensure consistency and improve the quality of the input data. Specifically, unnecessary newline characters were removed from the “body”, “subject”, and “answer” fields and replaced with spaces to maintain sentence continuity. Whitespace trimming was also performed to eliminate leading and trailing spaces. In addition, five tag columns were dropped. These columns provided granular topic classifications for each ticket, but only the first three were retained, since the remaining ones contained a high proportion of missing values and were likely to introduce noise or bias into the analysis.

After this initial preprocessing stage, the issue of missing values in the subject column became pivotal. Approximately **2,600** tickets lacked a subject (a concise summary of the customer’s inquiry). Instead of discarding these records, which still contained valuable information in the body field, a generative AI tool was employed to reconstruct the missing subjects. Specifically, the AI function in Google Sheets was used to automatically generate subject lines that were appropriate to the corresponding body text. Through this process, **2,604** ticket subjects were completed, preserving rows that otherwise would have been excluded and ensuring that potentially useful information was retained for subsequent analysis.

A similar strategy was also applied, at a smaller scale, to the “tag_2” and “tag_3” columns, where 73 missing values were completed using the same AI-based method. This additional step further reduced data sparsity in the retained tag columns, improving their suitability for downstream modeling tasks.

### 3.3 Dataset Characteristics and Schema

The final dataset used for this analysis is contained in a single comma-separated value (CSV) file named “Cleaned_Tickets.csv”. After preprocessing, the dataset consists of **16,337** records, each representing a unique support ticket, and **9** distinct features (columns). The schema of the dataset is detailed in Table 1 below. It comprised three primary text fields and six categorical fields providing structured metadata for each ticket.

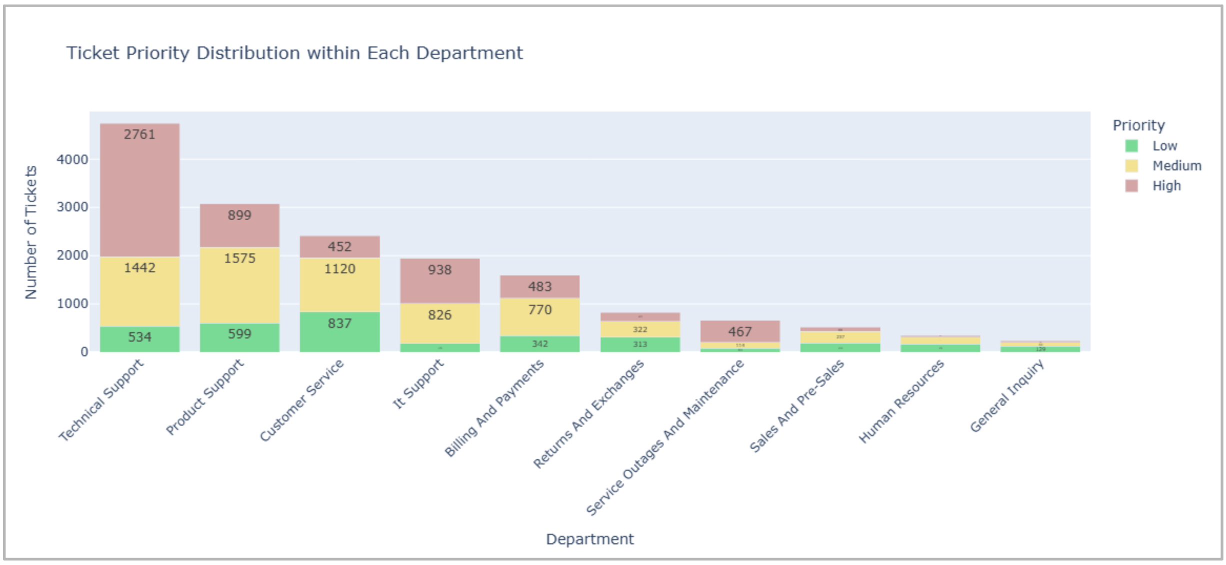
*Table 1: Data Description*

| | Feature Name | Type | Description | Example Values |
|---|---|---|---|---|
| 1 | subject | Text (String) | Summary of the customer’s ticket inquiry or issue | “Account Disruption”, “Feature Query”, “System Interruptions” |
| 2 | body | Text (String) | A detailed text of the ticket | “Dear Customer Support Team, I am writing to report a significant problem with the centralized account management portal…” |
| 3 | answer | Text (String) | The full text of the support agent’s response, providing information, or resolution steps | “Thank you for reaching out. We are aware of the outage affecting the centralized account management system…” |
| 4 | type | Categorical (Nominal) | The type of ticket | “Incident”, “Request”, |

| | | | | |
|---|---|---|---|---|
| 5 | department | Categorical (Nominal) | Specifies the department to which the email ticket is routed | “IT Support”, “Technical Support”, “Billing and Payments” |
| 6 | priority | Categorical (Ordinal) | Indicates the urgency and importance of the issue | “High”, “Medium”, “Low” |
| 7 | tag_1 | Categorical (Nominal) | Tags/categories assigned to the ticket | “Account”, “Product”, “Network” |
| 8 | tag_2 | Categorical (Nominal) | Tags/categories assigned to the ticket | “Disruption”, “Feature”, “Payment |
| 9 | tag_3 | Categorical (Nominal) | Tags/categories assigned to the ticket | “Documentation, “Performance”, “Outage” |

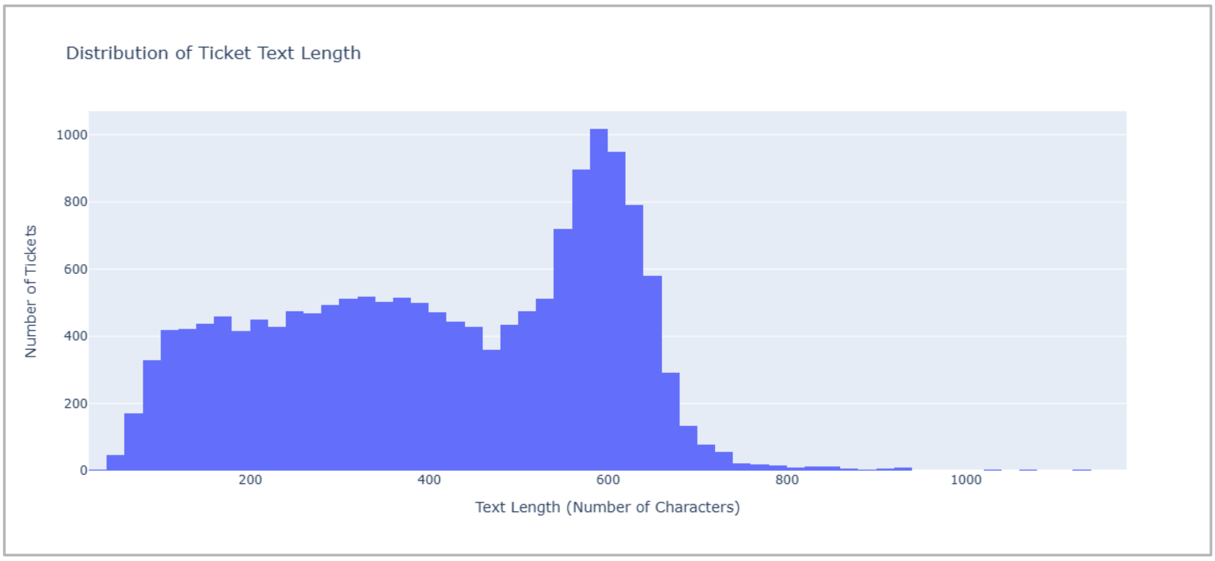
Following the structural definition of the dataset, visual exploration was undertaken in order to see our data better and uncover trends and patterns. First of all, Figure 1 illustrates the proportion of ticket priorities (Low, Medium, High) within each department. It can be observed that the “Technical Support” and “IT Support” departments exhibit a noticeably higher proportion of high-priority tickets compared to other departments, indicating that the issues handled by technical teams are often more critical in nature.

*Figure 1*



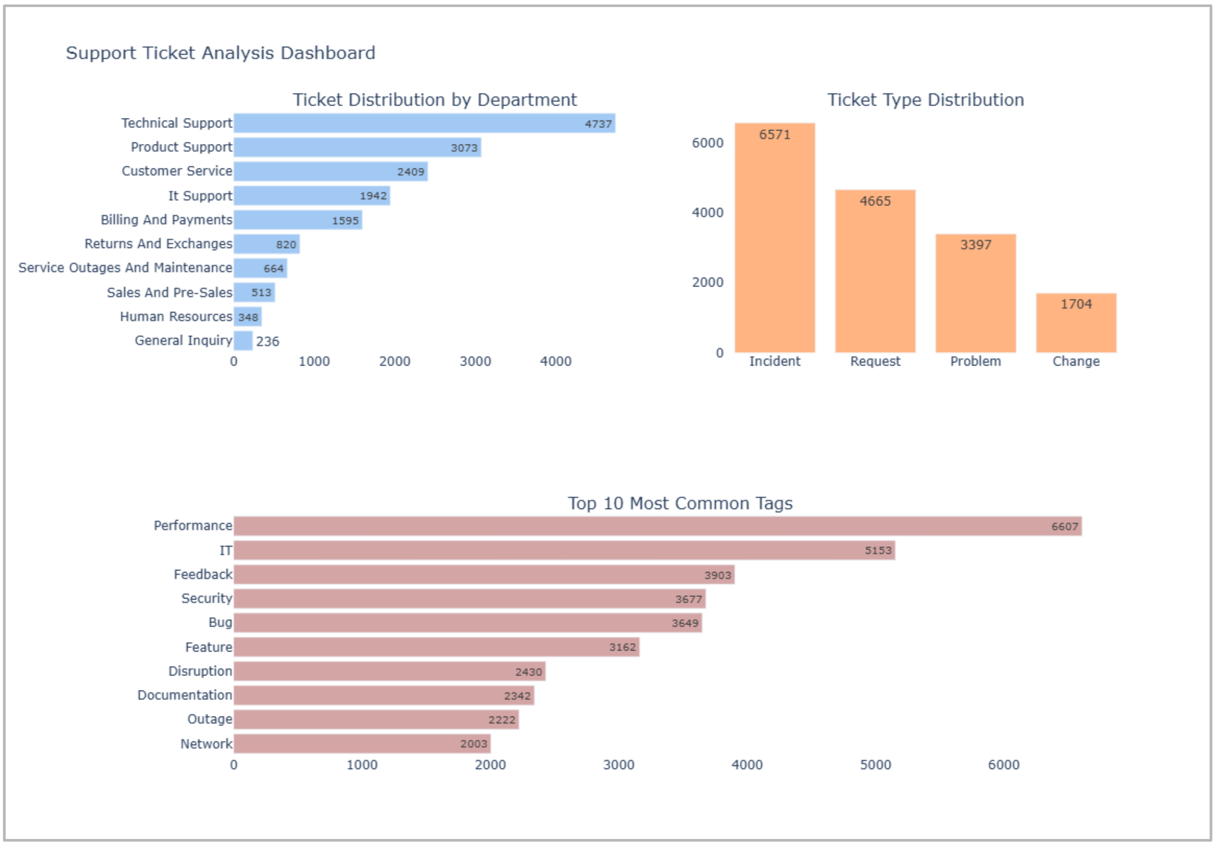
Furthermore, the histogram in Figure 2 presents the distribution of character lengths in the ticket bodies. The distribution is right-skewed, with a large concentration of tickets containing fewer than 500 characters. This suggests that most tickets consist of relatively short support requests. The extended tail to the right indicates that, although less common, some tickets include highly detailed and lengthy descriptions of issues.

*Figure 2*



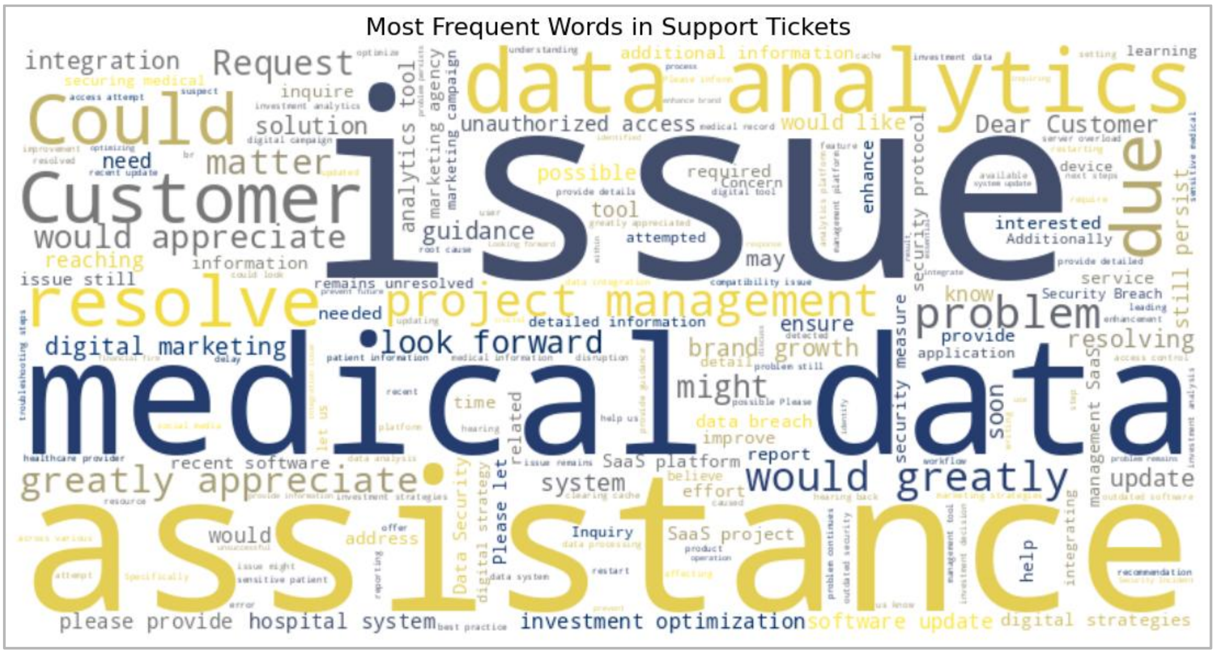
The dashboard in Figure 3 provides an overview of the customer support tickets. The bar chart on the left shows that the "Technical Support" department receives the highest volume of tickets. The second chart on the right indicates that the most frequent ticket type is "Incident", followed by "Request". Finally, the third chart reveals that the most common tags assigned to tickets relate to Performance, IT, and Feedback.

*Figure 3*



Finally, Figure 4 contains a word cloud. It is a visualization that represents the most frequently used words in the customer support tickets, with the size of each word corresponding to its frequency. Prominent words such as “issue,” “assistance,” “resolve,” and “request” stand out. This suggests that a significant portion of custom inquiries are related to issues with account access, questions about the platform’s functionality, and requests.

*Figure 4*



## 4. Four separate classification models

### 4.1 Department Classification Model

In modern customer-support operations, incoming tickets must be rapidly routed to the correct department so they can be resolved efficiently. In large organizations, the departments receive tickets with very different topics and urgencies. Manual triaging, where human agents inspect the ticket and assign it to a department, is slow, prone to human error, and does not scale to thousands of tickets per day. This section outlines the technical implementation and theoretical foundations of the models developed for the automated classification of customer support tickets. The main goal was to assign each incoming ticket to one of the ten predefined departments based only on the text of the request (“subject” and “description”).

To achieve this, two deep learning models were implemented and evaluated:

1. A baseline model built from scratch, based on embeddings and a Multi-Layer Perceptron (MLP).
2. A fine-tuned pretrained Transformer (BERT). This approach utilizes transfer learning to adapt a powerful pre-trained language model to the classification task, aiming for higher accuracy.

The following subsections will detail the theoretical concepts, architecture, and training strategies for each of these models.

A brief overview of the core concepts is necessary to understand the models' design and functionality.

### 4.1.1 Word Embeddings

Traditional approaches relied on Bag-of-Words or TF-IDF features, but these methods struggle with raw text, ignoring word order and context, which can be limiting in complex tasks like ticket classification. For the models implemented, word embeddings were utilized. In this method, each word is mapped to a point in a multi-dimensional space where semantically similar words are located closer to one another. Embeddings can either be:

- From scratch: the model learns on training data (randomly initialized and updated during training).
- Pre-trained: the model has already learned word meanings from huge text collections and then is adapted to a specific task.

### 4.1.2 Pretrained Transformers

Transformers such as BERT (Bidirectional Encoder Representations from Transformers) represent a major advance in NLP. BERT is trained on billions of words with a masked language modeling objective, allowing it to capture deep contextual information. Its key innovation is the self-attention mechanism, which enables the model to dynamically weigh the importance of different words in a sentence relative to each other. Unlike earlier models that processed text in one direction (left-to-right or right-to-left), BERT considers the entire sentence at once, allowing it to understand a word's meaning based on its full context.

### 4.1.3 Transfer Learning and Fine-Tuning

Transfer learning is a technique in which a model created for one task is utilized as the foundation for a model for another task. In NLP, this is taking a large pre-trained language model, such as BERT, and tailoring it to a specific downstream purpose. This procedure is known as fine-tuning. During fine-tuning, the pre-trained weights of the BERT model are not frozen but are updated with a very low learning rate.

### 4.1.4 Baseline Model

The first model was implemented as a simple and computationally efficient baseline to establish a performance benchmark. It learns word representations solely from the ticket data and uses a simple neural network to classify them. The architecture consists of the following layers:

- Embedding Layer: An embedding layer with 300 dimensions maps each token ID to a vector. The weights were initialized randomly and trained from scratch.
- Pooling: To handle variable-length tickets, the sequence of token embeddings was converted for each ticket into a single fixed-sized vector. This was achieved by taking the average of token embeddings, weighted by attention mask.
- MLP Classifier: The resulting average vector was fed into a two-layer MLP with a hidden dimension of 512 units. A ReLU activation function was used after the first layer, followed by a dropout layer (p = 0.3) for regularization. The final layer outputs logits for the 10 department classes.

The model contains approximately 8.9 million trainable parameters, making it significantly smaller than the BERT-based model. Prior to dividing the data into training (80%), validation (10%), and test (10%) sets, the data had to be prepared by combining text fields and encoding labels. Text was tokenized by converting it to integer IDs using the **BertTokenizer** vocabulary. A key aspect of the training was addressing the dataset's class imbalance by applying class weights to the **CrossEntropyLoss** function. The model was trained using the **AdamW** optimizer (learning rate: 1e-3) and an early stopping mechanism to save the best checkpoint based on the validation macro F1-score.

### 4.1.5 Fine-Tuning BERT Classifier

The second model is a more sophisticated approach that leverages a large, pre-trained language model for higher accuracy. This model uses transfer learning to adapt the contextual knowledge of the pre-trained **bert-base-case** model for ticket classification. The architecture consists of:

- The pre-trained model serves as the core component, processing the input text to generate contextualized embeddings.
- A Classification head attached to BERT's CLS token output. This head consists of a dropout layer (p = 0.3) and a single Linear layer that maps the BERT output to the 10 department classes.

This model has approximately 108 million parameters; the majority of them are pre-trained.

Finally, the implementation focused on fine-tuning the pre-trained model. To ensure a fair comparison, the data loading and splitting processes were identical to those used in the baseline model. A very low learning rate (2e-5) was used for fine-tuning with the **AdamW** optimizer. This is pivotal for transferring the pre-trained weights to the new model while preserving their learned knowledge. Similar to the baseline, the training loop used early stopping based on the validation weighted F1-Score to select the best-performing model (Appendix, Figure 5).

### 4.1.6 Results and Model Comparison

Both models were successfully trained and evaluated on the test set. The fine-tuned BERT model, as expected, outperformed the from-scratch baseline. The baseline model, while computationally efficient, achieved a weighted F1-score of **0.61** on the test set. The limitation of this model is that it struggles to capture deeper semantic and contextual meanings within the support tickets, particularly for classes that have significant semantic overlap or are under-represented, such as "General Inquiry" (F1-score of 0.48).

On the other hand, the fine-tuned BERT model achieved a great weighted F1-score of **0.73** on the test set. Since it was trained on a much larger dataset,  the model is more robust at distinguishing nuanced requests, leading to improved performance across all departments. In particular, it demonstrated a notable improvement in correctly identifying "General Inquiry" tickets (F1-score of 0.64) compared to the baseline. A summary of the key differences and final performance metrics is presented in Tables 3 & 4 (Appendix). Overall, the results clearly indicate that the knowledge transfer from a large pre-trained model offers a significant advantage over training a smaller model from scratch.

## 4.2 Tags Classification Model

Support tickets often contain multiple distinct issues that can be categorized using tags. Effective multi-label tagging is essential for granular analysis, identifying trends, and ensuring that all facets of a customer's request are addressed. Manually assigning multiple tags is time-consuming and inconsistent. This section details the development of two models for multi-label tag prediction, where each ticket can be assigned one or more relevant tags from a predefined set.

The project's objective was to create a system capable of accurately predicting multiple tags based on the ticket's text. The following models were developed and compared:

1. A baseline model built from scratch, using learned word embeddings and a simple yet effective MLP architecture.
2. A fine-tuned pretrained Transformer (DeBERTa-v3), leveraging a state-of-the-art language model to achieve superior performance in understanding the complex nuances of ticket text.

The subsequent sections will provide an in-depth look at the theoretical underpinnings, architectural design, and training methodologies for both models.

To understand the models, a brief overview of the core concepts is necessary.

### 4.2.1 Multi-Label Classification

Unlike multi-class classification, where each input belongs to a single category (like "Department"), multi-label classification allows for an input to be assigned to zero or more categories simultaneously. This is a more complex task as the model must learn to identify multiple independent patterns within the same text. The final activation layer in such models typically uses a Sigmoid function, which outputs an independent probability for each potential label, rather than a Softmax function which would force the probabilities to sum to one.

### 4.2.2 Pretrained Transformers (DeBERTa)

DeBERTa (Decoding-enhanced BERT with disentangled attention) represents an evolution of the original BERT model. Its key innovation is the disentangled attention mechanism, which encodes words using two separate vectors for content and relative position. This allows the self-attention mechanism to better capture the dependencies between words based not just on their meaning but also on their proximity to each other. For tasks requiring a deep understanding of language structure, like ticket classification, DeBERTa-v3 often provides a significant performance boost.

### 4.2.3 Threshold Tuning

In multi-label classification, the model outputs a probability score (from 0 to 1) for each tag. A decision threshold is required to determine whether to assign a tag. A standard threshold of 0.5 is often suboptimal. Threshold tuning is a critical post-training step where the optimal probability cutoff is determined for each tag individually by evaluating performance (typically F1-score) on a validation set. This is especially important in datasets with imbalanced tag distributions, as some rare tags may require a lower threshold for effective prediction.

### 4.2.4 Model 1: From-scratch Baseline Model

The baseline model was designed to be a simple, computationally efficient benchmark, learning tag relationships directly from the provided ticket data. Its architecture is as follows:

- **Embedding Layer**: This layer maps token IDs to dense vectors of 256 dimensions. While a pretrained BertTokenizer was used to convert text to token IDs based on its established vocabulary, the embedding layer itself was not pretrained. Its weights were randomly initialized and then trained from scratch on the ticket dataset.
- **Weighted Pooling**: To create a single fixed-size representation for each ticket, a weighted average of the token embeddings was calculated, using the attention mask to ensure padding tokens were ignored.
- **MLP Classifier**: The resulting vector was passed to a two-layer MLP with a hidden dimension of 512 units, a ReLU activation, and a dropout layer (p=0.1) for regularization. The final layer outputs logits for all possible tag classes.

The model was trained using the AdamW optimizer (learning rate: 1e-3) and a ReduceLROnPlateau scheduler to adjust the learning rate based on the validation F1-score. To combat the severe class imbalance inherent in the tag data, a WeightedRandomSampler was used during training. This sampler gives a higher probability of selecting tickets with rarer tags, ensuring the model gets adequate exposure to all classes. The best model was saved based on early stopping with a patience of 2 epochs on the validation micro F1-score.

### 4.2.5 Model 2: Fine-Tuning DeBERTa-v3 Classifier

The second, more advanced model leverages the powerful DeBERTa-v3-large pretrained transformer to achieve higher accuracy. Its architecture consists of:

- **The DeBERTa-v3-large Model**: This serves as the core feature extractor, generating highly contextualized representations of the input text.
- **Classification Head**: A standard sequence classification head is attached to the DeBERTa output. It consists of a pooler and a single Linear layer that maps the transformer's output to the logits for each tag class.

The fine-tuning process was carefully configured to adapt the pretrained model to this specific task. A low learning rate of 2e-5 was used with the AdamW optimizer, along with a linear scheduler with warmup. As with the baseline, the best model was saved using an early stopping mechanism based on the validation micro F1-score. Following training, a crucial threshold-tuning step was performed on the validation set to determine the optimal probability cutoff for each individual tag, maximizing the F1-score on a per-tag basis.

### 4.2.6 Results and Model Comparison

Both models were trained and evaluated on the held-out test set. The results clearly demonstrate the superiority of the fine-tuned DeBERTa-v3 model for this complex multi-label task.

The from-scratch baseline model achieved a respectable micro F1-score of 0.65 on the test set. While fast to train, its performance was limited by its inability to learn from a massive corpus of text, making it struggle with the nuanced and often overlapping terminology present in support tickets. Its macro F1-score was low (0.12), indicating difficulty in predicting rare tags despite the use of a weighted sampler.

The fine-tuned DeBERTa-v3 model, benefiting from transfer learning, achieved a significantly higher micro F1-score of 0.76 on the test set. The model's deep contextual understanding allowed it to excel at identifying multiple relevant tags within a single ticket.

## 4.3 Types Model

In IT service management, one of the most critical steps in ticket triage is identifying the type of request. Service tickets are generally categorized into four standard types: Change, Incident, Problem, and Request. Correctly distinguishing between these ensures that workflows follow the right ITIL procedures: for example, Problems often require root cause analysis, while Incidents demand immediate resolution. Manual classification is error-prone and costly, particularly at scale. Automating this step with machine learning accelerates response times and improves consistency.

This section details the implementation of two deep learning approaches for ticket type prediction. The first is a lightweight baseline model leveraging frozen Transformer embeddings, while the second is a fine-tuned DistilBERT model designed to capture deeper semantic nuances in the text. Both approaches aim to predict the correct type based on the combined subject and body of the ticket. To understand the motivation behind the two models, the following concepts are relevant.

### 4.3.1 Ticket Classification as a Multi-Class Task

Unlike tags, which are multi-label, the type task is multi-class classification, where each ticket must belong to exactly one category. The models therefore use a Softmax activation in the final layer, producing a probability distribution over the four possible types. Training is performed with a cross-entropy loss, optionally weighted to account for imbalanced class frequencies.

### 4.3.2 Transformers and DistilBERT

Transformers are now the standard for natural language processing. DistilBERT is a distilled variant of BERT, offering 97% of BERT's performance with ~40% fewer parameters. This makes it a suitable trade-off between accuracy and computational efficiency for production settings.

The models here explore two strategies:

- Using DistilBERT as a frozen feature extractor (embeddings are mean-pooled and fed into a simple classifier).
- Fine-tuning the entire encoder with a small learning rate, allowing the model to specialize in the support ticket domain.

### 4.3.3 Class Imbalance and Problem Boost

The dataset showed a notable imbalance, with Problem tickets under-represented compared to Incidents and Requests. To address this, class-weighted cross-entropy loss was employed. In addition, the weight for the Problem class was manually boosted by 15%, improving recall for this under-predicted category.

### 4.3.4 Model 1: Frozen DistilBERT with Linear Head

The baseline model was implemented as a computationally efficient benchmark. Its architecture is as follows:

- Frozen DistilBERT Encoder: The pre-trained encoder provides contextualized embeddings for each token, but its parameters remain fixed.
- Mean Pooling: Token embeddings are aggregated via attention-mask–weighted mean pooling to produce a fixed-size sentence representation.
- Classification Head: A dropout layer (p = 0.2) followed by a single linear layer maps the pooled embedding to logits for the four ticket types.

Training used class-weighted cross-entropy loss with the “Problem boost” and the AdamW optimizer (learning rate = 2e-4). Early stopping was applied based on the validation weighted F1-score.

This model provides a fair baseline, learning only the classifier head while keeping the large encoder frozen.

### 4.3.5 Model 2: Fine-Tuned DistilBERT

The second model allows the full encoder to update during training, with differential learning rates to carefully balance stability and adaptation:

- Encoder Parameters: Fine-tuned with a very low learning rate (2e-5) and weight decay (0.01).
- Classification Head: Trained with a slightly higher learning rate (1e-4) to adapt more quickly.
- Scheduler and Regularization: A linear warm-up scheduler (6% steps), gradient clipping (max norm = 1.0), and mixed-precision training were used for stable convergence.

This configuration enables the model to adapt the pre-trained representations to the domain of customer support tickets while preserving general linguistic knowledge.

### 4.3.6 Results and Model Comparison

Both models were trained and evaluated on an 80/20 stratified split of the data. Performance metrics are summarized below.

*Table 2: Type Models Comparison*

| Metric | Frozen Model | Fine-Tuned Model | Change |
|---|---|---|---|
| Accuracy | 0.76 | 0.86 | +0.10 |
| Macro Avg Precision | 0.74 | 0.87 | +0.13 |
| Macro Avg Recall | 0.76 | 0.88 | +0.12 |
| Macro Avg F1-Score | 0.76 | 0.88 | +0.12 |
| Weighted Avg F1-Score | 0.76 | 0.86 | +0.10 |
| Problem Recall | 0.46 | 0.78 | +0.32 |
| Incident Recall | 0.74 | 0.76 | +0.02 |
| Change Recall | 0.90 | 1.00 | +0.10 |
| Request Recall | 0.94 | 0.99 | +0.05 |

The results show that fine-tuning DistilBERT yields a +12% improvement in macro F1 compared to the frozen baseline. The largest gain came from the Problem class, where recall jumped by 32 percentage points, resolving a major shortcoming of the baseline model.

Confusion matrices further revealed that the majority of residual misclassifications occur between Incident and Problem, reflecting their semantic overlap in real-world support tickets. Nonetheless, Change and Request predictions were nearly perfect.

Overall, the fine-tuned DistilBERT model clearly outperformed the baseline, offering state-of-the-art performance for type classification and significantly reducing misclassification of under-represented categories.

## 4.4 Priority Classification Model

In customer support operations, it is important to prioritize the importance of each ticket based on its urgency, which needs to be handled. A critical case that disrupts the daily operation of a service/product needs to be addressed immediately, while less significant cases can be dealt with later without major impact. Relying on manual triage, where support agents read each ticket and decide its urgency, is inefficient, error-prone, and difficult to sustain at a larger scale, where thousands of tickets are received each day

This section describes the models developed to automate priority prediction. The goal was to classify each ticket into one of three predefined levels: high, medium, or low, using only the text contained in the “subject” and “body” fields.

To achieve this, two deep learning models were implemented and evaluated:

1. A baseline model built from scratch, which learns word embeddings directly from the training data and applies a lightweight classifier.
2. A fine-tuned pretrained Transformer (DistilBERT), which adapts a large language model to the specific task of predicting ticket priority.

The following subsections highlight the design and training strategies for each model and explain how they address the challenges of automating ticket prioritization.

### 4.4.1 Baseline Model

The baseline model was implemented to establish a simple but effective starting point for priority classification. Its architecture was inspired by the FastText approach, emphasizing efficiency over complexity.

Each ticket was first tokenized into words, which were mapped to randomly initialized embedding vectors of 128 dimensions. These embeddings were not pre-trained but instead learned entirely from the ticket dataset during training. To obtain a fixed-length representation for each ticket, the embeddings of all words were averaged into a single vector.

This vector was then passed through a lightweight neural network consisting of a dropout layer for regularization and a fully connected output layer. The model produced logits corresponding to the three possible priority levels. Training was carried out using the AdamW optimizer with a learning rate of 3e-3, a batch size of 64, and early stopping based on the validation F1-score.

While relatively small in scale, this model served as a useful benchmark. It demonstrated how a from-scratch embedding approach can capture meaningful patterns in the ticket data, but also highlighted the limitations of simpler architectures when compared to more sophisticated Transformer-based methods.

### 4.4.2 Fine-Tuned Transformer Model

To overcome the limitations of the baseline model, a more advanced approach was implemented using a pre-trained Transformer model. Specifically, DistilBERT was selected, providing a balance between computational efficiency and strong language understanding. DistilBERT is a distilled version of BERT, trained on large text corpora to capture contextual meaning and relationships between words.

For this task, the pre-trained model was fine-tuned to predict ticket priority levels. Input tickets were cleaned and tokenized using the DistilBERT tokenizer, which splits text into subword units and assigns each a numerical ID. These sequences were padded or truncated to a fixed maximum length, then fed into the model.

In addition to DistilBERT’s output, a classification head was added, consisting of a dropout layer with a dropout rate of 0.3 and a linear layer that maps the CLS token representation to the three priority classes. Unlike the baseline, the weights of the pre-trained layers were not frozen; they were updated during training with a very low learning rate (2e-5) to adapt the model’s general knowledge to the specifics of the ticket dataset.

Training used the AdamW optimizer with weight decay for regularization, a batch size of 8, and a ReduceLROnPlateau scheduler to adjust the learning rate when validation performance plateaued. Early stopping was again applied, saving the best checkpoint based on the validation F1-score.

This fine-tuned model proved significantly more effective than the baseline, as it leveraged the linguistic knowledge embedded in the pre-trained weights. Its ability to capture subtle contextual cues in tickets made it more robust in distinguishing between priority levels, especially in cases where wording was ambiguous or classes were imbalanced.

### 4.4.3 Results and Model Comparison

Both models were trained and evaluated using an identical 80/10/10 split of the dataset. The baseline model, trained from scratch with randomly initialized embeddings, steadily improved across epochs but plateaued with a weighted F1-score of 0.57 on the held-out test set. Performance varied across classes: the model achieved an F1 of 0.62 for High priority, but only 0.43 for Low priority, revealing challenges in handling underrepresented classes. Overall, the baseline provided a computationally efficient benchmark, but its reliance on embeddings learned from limited data limited its ability to capture deeper contextual nuances.

The fine-tuned Transformer model (DistilBERT) demonstrated a clear performance advantage. By leveraging pre-trained contextual representations, it achieved a weighted F1-score of 0.73 on the test set, representing a substantial gain over the baseline. Improvements were consistent across classes: High priority tickets reached an F1 of 0.78, Medium priority 0.72, and even the weaker Low priority class improved to 0.64. These results highlight the model's strength in distinguishing subtle differences in ticket language and in coping with class imbalance.

In summary, while the baseline model established a useful point of comparison with modest computational requirements, the fine-tuned Transformer proved far more effective for the priority classification task. The results demonstrate the value of transfer learning: knowledge distilled from large-scale pre-training transferred successfully to the more specialized domain of customer support tickets. A summary of the key differences and final performance metrics is presented in Table 5 (Appendix).

## 5. Multi Model

In addition to training individual models for each prediction task (type, department, priority, and tags), a joint multi-task model was implemented. This approach uses a single shared encoder (DistilBERT) to process the input text (ticket subject and body), followed by four separate classification heads corresponding to the four tasks.

The training followed a two-stage procedure:

1. Baseline (frozen encoder): The encoder weights were kept frozen, and only the task-specific heads were trained. This established a baseline performance level and ensured that the heads were functioning correctly.

2. Fine-tuning: In the second stage, the top layers of the encoder were unfrozen and trained jointly with the heads using differential learning rates (lower for the encoder, higher for the heads). This allowed the encoder to adapt to the domain-specific ticket data while preventing catastrophic forgetting.

The baseline model achieved a macro F1 of ~0.80 for type classification, ~0.26 for department, ~0.41 for priority, and a micro F1 of ~0.60 for tags. After fine-tuning, performance improved across all tasks: type classification reached 0.83 macro F1, department increased to 0.31 macro F1, priority improved to 0.44 macro F1, and tags achieved 0.62 micro F1. These results confirm that fine-tuning the shared encoder provided meaningful gains, particularly for the more difficult department and priority tasks.

Compared to the individual models, the multi-task approach has the advantage of parameter sharing: the encoder learns representations that benefit multiple tasks simultaneously, which is computationally efficient and can improve generalization. The trade-off is that rare labels, especially in the department and tags tasks, remain difficult due to strong class imbalance. Nonetheless, the multi-task model demonstrates that a shared architecture can effectively handle multiple classification objectives in parallel while maintaining competitive performance with the individual models.

Although the multi-task model demonstrated that a shared encoder could jointly learn the four classification tasks, the trade-offs in performance were significant. While type prediction remained very strong, both department and priority classifications suffered from class imbalance and overlapping semantics, and multi-label tag prediction continued to favor frequent labels while neglecting rare ones. More importantly, the joint architecture introduced coupling between the tasks, meaning that optimization for one objective could come at the expense of another. For these reasons, the final system design opted to proceed with the individual single-task models, which delivered more stable and interpretable performance per category. This choice also aligned better with the requirements of the agentic workflow: by exposing each classifier independently as part of the predict_ticket_attributes tool, the ReAct agent could orchestrate predictions in a modular fashion, ensuring that each attribute (type, department, priority, tags) was predicted by its specialized model. This modularity improved transparency, allowed easier debugging and replacement of individual models, and ultimately resulted in a more reliable pipeline for the end-to-end autonomous ticket processing system.

## 6. Agentic Workflow Implementation

While the machine learning models provide the core intelligence for classification, the ultimate goal of this project was to create a fully autonomous system that could execute a multi-step business process. This required moving beyond simple prediction and implementing an agentic workflow. This section details the architecture and implementation of the AI agent responsible for orchestrating the entire ticket processing pipeline, from initial text cleaning to the final creation of a task in an external system.

The system is architected around a central ReAct agent, which acts as the "brain" of the operation. This agent uses a predefined set of tools to interact with the world and execute its tasks. The entire workflow is powered by a local Large Language Model (LLM) for reasoning, ensuring privacy and control.

### 6.1 The ReAct Agent

The core of the system is a goal-oriented agent built using the LangChain framework. The chosen methodology was ReAct, which stands for "Reasoning and Acting." This framework enables the agent to synergize its reasoning capabilities with action-oriented tool use. The agent operates in a loop, observing a situation, thinking about the next logical step, and then taking an action by calling one of its available tools. This Thought -> Action -> Observation loop allows the agent to break down a complex goal into a sequence of manageable steps, handle intermediate results, and proceed logically toward a final answer.

- **Prompt Engineering**: The agent's behavior is strictly governed by a carefully constructed system prompt. This prompt defines the agent's persona, its ultimate goal, the tools it has access to, and, most importantly, the exact sequence of actions it must follow: 1) clean_text, 2) predict_ticket_attributes, and 3) create_clickup_task. This strict instruction set, a key part of prompt engineering, ensures the workflow is reliable and predictable, preventing the agent from deviating from the required business process.
- **Local LLM**: The reasoning engine for the agent is a local Large Language Model (llama3) served via Ollama. This design choice provides significant benefits, including data privacy (as no ticket information is sent to a third-party API), zero cost for inference, and complete control over the model environment.

## 6.2 Tool Design and Implementation

The agent's capabilities are exclusively defined by the tools provided to it. Three distinct tools were developed to grant the agent all the functions necessary to complete its goal.

1. **clean_text**: This tool provides a deterministic way to preprocess the raw ticket subject and body. It handles tasks like stripping signatures and correcting common typos. By encapsulating this logic in a tool, the agent can initiate the workflow with a clean, standardized input.
2. **predict_ticket_attributes**: This tool serves as a critical interface, or "facade," to the complex, multi-model ML pipeline developed in the previous stages. The agent calls this single tool to get all four predictions (Tags, Department, Type, and Priority). This design follows the principle of separation of concerns: the agent doesn't need to know about the four different underlying models or how they are orchestrated. Furthermore, this tool was specifically designed to return a concise summary of the predictions, stripping out the verbose confidence scores to keep the agent's context clean and prevent the reasoning process from being disrupted by extraneous data.
3. **create_clickup_task**: This tool handles all interaction with the external ClickUp API. It takes the cleaned text and the AI-generated predictions and creates a new, fully detailed task. The tool is responsible for mapping predicted labels to the correct custom fields, setting the priority, and adding tags. It also includes a one-second delay after task creation to prevent potential API race conditions, making the external integration more robust.

## 6.3 Demonstration and User Interface

To provide an interactive and user-friendly way to showcase the end-to-end agentic workflow, a simple web interface was built using Streamlit. This UI allows a user to input a ticket subject and body and, with a single click, initiate the entire process. The application provides real-time feedback while the agent is running and displays the final JSON result, including a direct link to the newly created task in ClickUp. For an improved user experience, the ML models are pre-loaded when the application starts, ensuring that the agent's processing is responsive. This interface serves as the primary method for demonstrating the project's success locally.

## 7. Conclusion

This project successfully demonstrated the capabilities of an integrated AI system in modernizing customer support operations. By developing and comparing multiple machine learning models, it was determined that fine-tuned Transformer architectures like BERT, DeBERTa, and DistilBERT consistently outperform simpler, from-scratch baselines for classifying ticket department, tags, type, and priority. Aside from simple classification, the implementation of a ReAct agent orchestrating these models through a set of specialized tools highlights the feasibility of creating a fully autonomous, end-to-end triaging pipeline powered by a local Large Language Model. This AI-driven system directly addresses the core business needs of reducing manual triage, minimizing response times, and cutting operational costs. Finally, this leads to increased customer satisfaction and engagement.

Firstly, several technical challenges were encountered and overcome. A primary difficulty was the class imbalance present in the department and tags datasets, where a few categories dominated the data. While techniques like class weighting and weighted random sampling proved effective for the baseline models, the fine-tuned Transformer models still faced difficulties with rare labels, a common issue in multi-label classification. Another significant challenge was the hyperparameter tuning for the fine-tuning process.

In addition, identifying the optimal learning rate and scheduler settings for large models like DeBERTa required careful experimentation; a higher learning rate could negatively affect the pre-trained weights. Finally, designing the multi-task model introduced complexity in balancing the four distinct loss functions, where optimizing for one task sometimes came at the expense of another, leading to the decision to use specialized single-task models for the final agentic workflow.

Last but not least, several aspects exist for enhancing the system. The models could be further improved by using more sophisticated techniques for handling class imbalance, such as experimenting with different loss functions. Additionally, the agentic workflow could be expanded with new tools, allowing it to not only triage tickets but also to retrieve information from a knowledge base or even generate draft responses. This project provides a solid proof-of-concept, setting the basis for the creation of a full, AI-powered customer service automation platform.

## 8. Appendix

*Figure 5: Training and validation loss and accuracy curves for the fine-tuned BERT model over 31 epochs.*

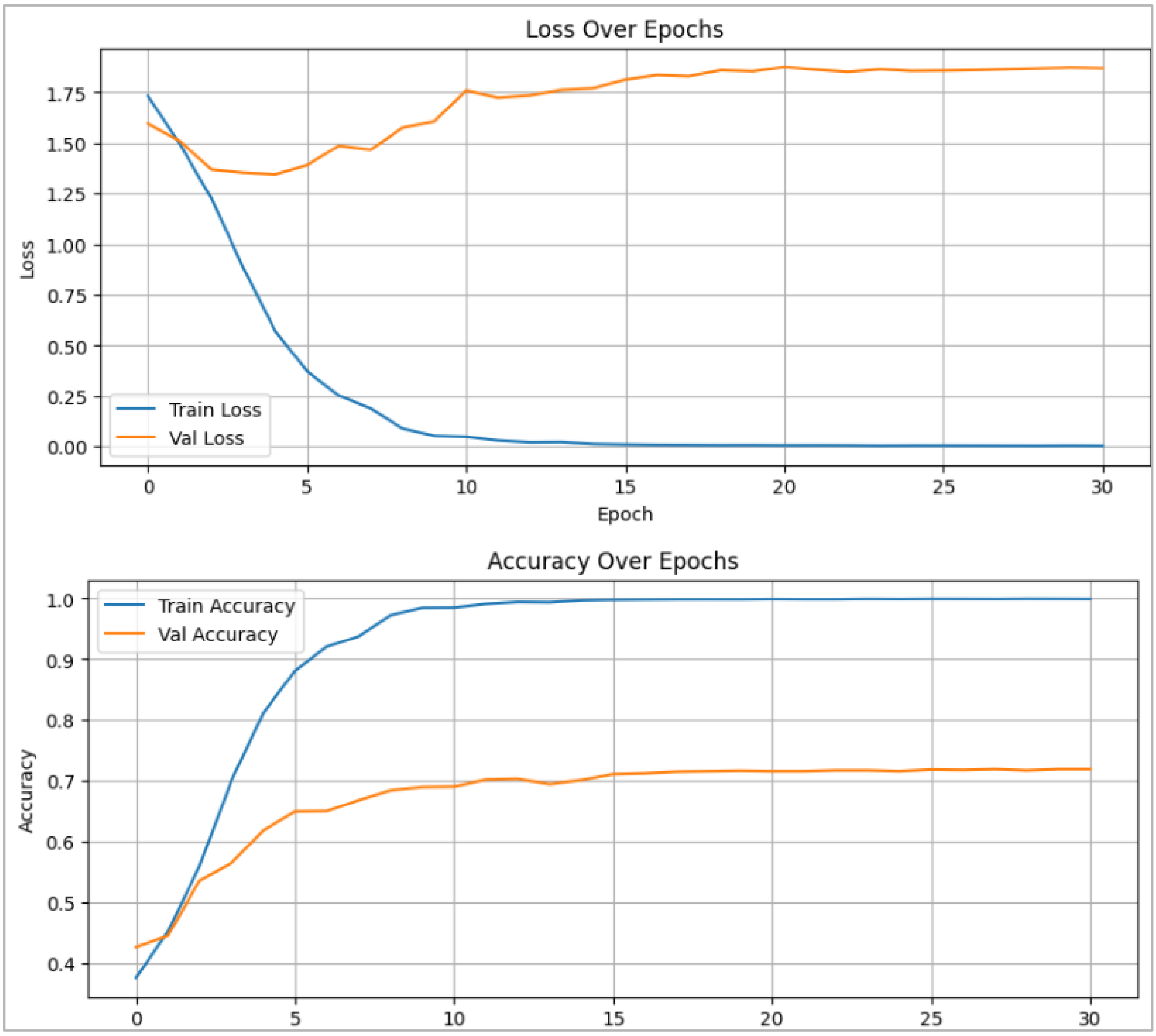


*Table 3: Department Models Comparison*

| Feature | Model 1 (Baseline) | Model 2 (Fine-Tuned Bert) |
|---|---|---|
| Approach | Training from scratch | Transfer Learning/Fine-Tuning |
| Parameters | ~8.9 Million | ~108.3 Million |
| Learning Rate | 1e-3 | 2e-5 |
| Test Accuracy | 61.5% | 73.4% |
| Test F1 (Weighted) | 0.614 | 0.733 |

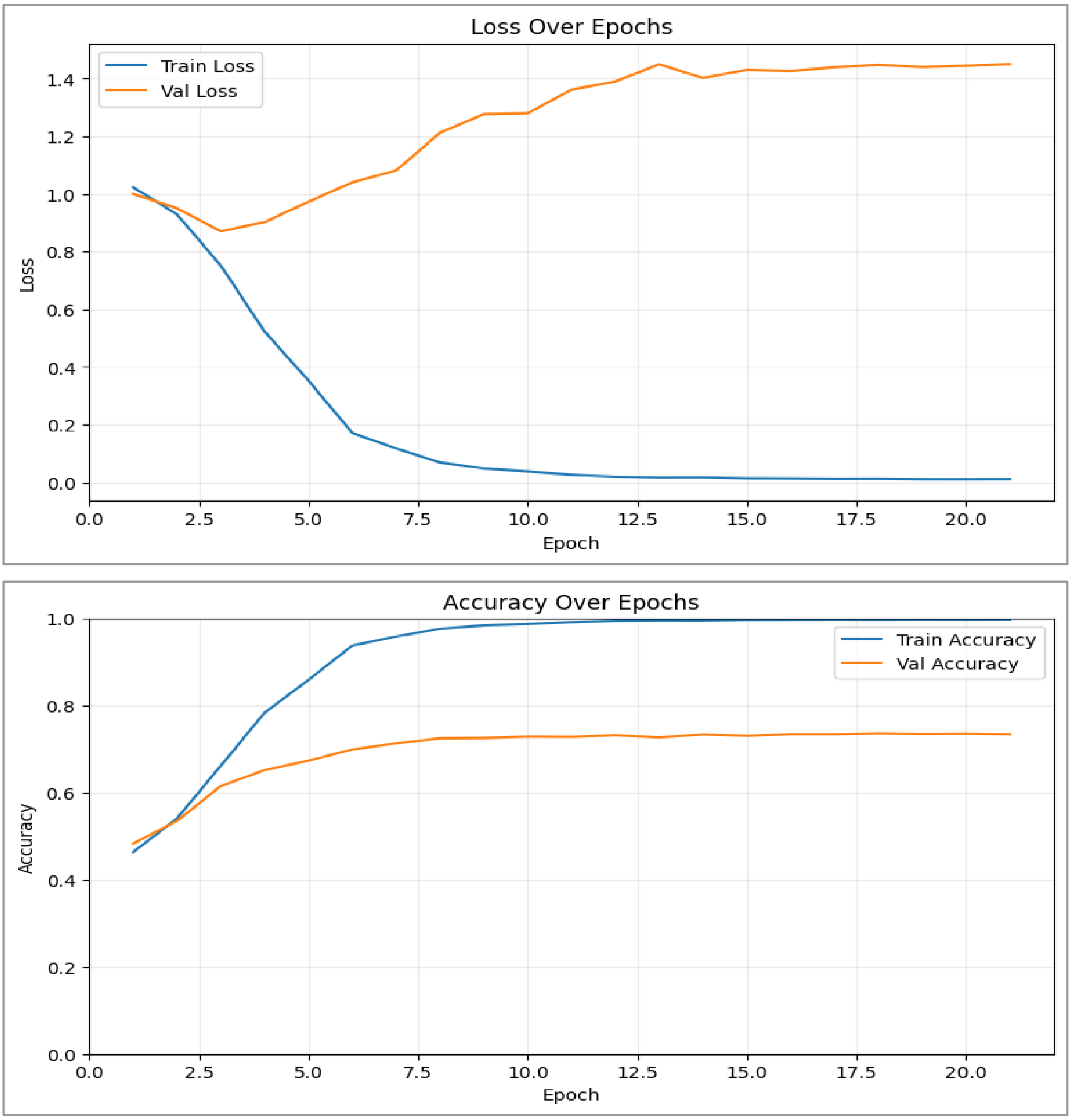
*Table 4: Comparison of test set performance metrics for the baseline (Model 1) and fine-tuned BERT (Model 2) classifiers.*

| Class | Model 1: Precision | Model 1: Recall | Model 1: F1-Score | Model 2: Precision | Model 2: Recall | Model 2: F1-Score | Support |
|---|---|---|---|---|---|---|---|
| Billing and Payments | 0.81 | 0.84 | 0.82 | 0.86 | 0.86 | 0.86 | 160 |
| Customer Service | 0.58 | 0.59 | 0.58 | 0.67 | 0.7 | 0.68 | 241 |
| General Inquiry | 0.43 | 0.54 | 0.48 | 0.7 | 0.58 | 0.64 | 24 |
| Human Resources | 0.51 | 0.6 | 0.55 | 0.75 | 0.69 | 0.72 | 35 |
| IT Support | 0.57 | 0.53 | 0.55 | 0.71 | 0.65 | 0.68 | 194 |
| Product Support | 0.61 | 0.52 | 0.56 | 0.74 | 0.69 | 0.71 | 307 |
| Returns and Exchanges | 0.5 | 0.65 | 0.57 | 0.78 | 0.74 | 0.76 | 82 |
| Sales and Pre-Sales | 0.52 | 0.51 | 0.51 | 0.62 | 0.63 | 0.62 | 51 |
| Service Outages | 0.65 | 0.62 | 0.64 | 0.84 | 0.77 | 0.8 | 66 |
| Technical Support | 0.64 | 0.66 | 0.65 | 0.73 | 0.79 | 0.76 | 474 |
| Accuracy |  |  | 0.62 |  |  | 0.73 | 1634 |
| Macro Avg | 0.58 | 0.61 | 0.59 | 0.74 | 0.71 | 0.72 | 1634 |
| Weighted Avg | 0.62 | 0.62 | 0.61 | 0.73 | 0.73 | 0.73 | 1634 |

*Table 5: Priority Models Comparison*

| Feature | Model 1 (FastText baseline) | Model 2 (DistilBERT fine-tuned) |
|---|---|---|
| Model | FastText baseline | DistilBERT fine-tuned |
| Learning Rate | 3e-3 | 2e-5 |
| Test Accuracy | 0.56 | 0.73 |
| F1 (Weighted) | 0.56 | 0.73 |
| Precision (Weighted) | 0.56 | 0.73 |
| Recall (Weighted) | 0.56 | 0.73 |
| F1 (Macro) | 0.54 | 0.71 |
| Precision (Macro) | 0.55 | 0.74 |
| Recall (Macro) | 0.53 | 0.70 |

*Figure 6-7: Training and validation loss and accuracy curves for the fine-tuned DistilBERT model over 21 epochs.*


Loss Over Epochs
Train Loss
Val Loss
Loss
Epoch
Accuracy Over Epochs
Train Accuracy
Val Accuracy
Accuracy
Epoch