Journal of Intelligent Decision and Computational Modelling

www.jidcm.reapress.com

J. Intell. Decis. Comput. Model. Vol. 2, No. 2 (2026) 137–154.

Paper Type: Original Article

Adaptive Transformer Learning for Robust Intent Classification and Slot Filling

Saeed Biabani Mahalli[1], Nastaran Zarafshan[2], Jamshid Pirgazi[1,*]

[1] Department of Electrical and Computer Engineering, University of Science and Technology of Mazandaran, Iran; s.biabanim@mazust.ac.ir; j.pirgazi@mazust.ac.ir.

[2] Department of Computer Engineering, Shahrood University of Technology, Semnan, Iran; nstrn.zarafshan@gmail.com.

Citation:

Received: 25 November 2025 *Revised: 14 February 2026* *Accepted: 09 April 2026*	Biabani Mahalli, S., Zarafshan, N., & Pirgazi, J. (2026). Adaptive transformer learning for robust intent classification and slot filling. *Journal of intelligent decision and computational modelling, 2*(2), 137-154.

Abstract

This study proposes a lightweight transformer-based model for intent classification and slot filling in conversational systems. By combining Multi-Head Attention (MHA), Root Mean Square (RMS) normalization, Swish Gated Linear Unit (SwiGLU) feedforward layers, and two-stage adaptive training on hard samples, the model improves robustness, handles imbalanced data, and achieves stronger accuracy and generalization in Natural Language Understanding (NLU) tasks.

Keywords: Intent classification, Slot filling, Transformer network, Multi-head attention, Imbalanced data, Deep learning.

1 | Introduction

In recent years, advancements in Natural Language Processing (NLP) have led to the development and improvement of human-machine interaction systems. Among these systems, intelligent assistants and conversational chatbots play a key role in enhancing user experience and increasing the efficiency of everyday processes. The performance of these systems depends on their ability to understand and interpret natural language, which involves two main tasks: intent classification, which identifies the purpose behind a user's request, and slot filling, which extracts the key information needed to fulfill the request. These tasks enable Natural Language Understanding (NLU) systems to accurately interpret user inputs and provide appropriate responses. Earlier approaches to intent classification and slot filling relied on conventional machine learning techniques such as Support Vector Machines (SVMs), Naive Bayes, and Hidden Markov Models (HMMs). These models depended heavily on labeled datasets and manually engineered features extracted from the input text. While these methods laid the groundwork for NLU, their performance was often limited by the quality of the handcrafted features and their inability to capture complex linguistic patterns and long-range dependencies in the data. Early research, such as studies by Tur et al. [1] and Ray and Craven [2], demonstrated that combining HMM and Conditional Random Fields (CRF) could deliver satisfactory performance in slot filling. Additionally, other models, such as the integration of semantic grammars with machine learning techniques, were proposed and utilized in early conversational systems like Speech Interpretation and

Corresponding Author: j.pirgazi@mazust.ac.ir

https://doi.org/10.48314/jidcm.v2i2.91

Recognition Interface (SIRI) [3]. However, these models faced scalability and performance limitations, particularly when handling complex data and linguistic diversity.

With the advancement of deep learning, models based on Recurrent Neural Networks (RNNs) entered this field. These models, due to their unique ability to process long sequences, proved to be highly suitable for natural language-related tasks. Research by Yao et al. [4] demonstrated that using Long Short-Term Memory (LSTM) for combining intent classification and slot filling could lead to significant improvements. These models were particularly effective for languages with complex and non-linear structures. Additionally, Seq2Seq models with Encoder-Decoder Architectures (EDA), especially in studies by Goo et al. [5] and Zhang and Wang [6], were introduced as a comprehensive approach for these tasks. By incorporating the attention mechanism Vaswani et al. [7], these models were able to address the issue of information loss in long sequences.

Pre-trained models such as [8–10] revolutionized NLP through transfer learning. These models, pre-trained on vast amounts of textual data and fine-tuned for specific tasks, delivered unprecedented performance across a wide range of NLP tasks. The model by Chen et al. [11] is one of the prominent approaches in combining intent classification and slot filling tasks, designed using the Bidirectional Encoder Representations from Transformers (BERT) architecture. This model leverages the powerful features of transfer learning and a dual-part attention-based architecture to simultaneously process semantic and syntactic information. In Joint BERT, input tokens are first processed by transformer layers, and then the output corresponding to the Classification Token (CLS) is used for intent classification, while the outputs associated with other tokens are used for slot identification. This approach enables interaction between the two tasks, thereby improving performance in both areas. In addition to BERT, models such as Yang et al. [12] and Raffel et al. [13] have also been employed in various NLP tasks, demonstrating a deeper understanding of semantic relationships and the ability to process incomplete data. These models exhibit a unique capability in handling noisy data and unstructured text. Furthermore, multilingual models like BERT and those of Conneau et al. [14] have enabled applications in low-resource languages and data-scarce scenarios. While transformer-based models have achieved remarkable success in NLP tasks, one of the main challenges in this field is maintaining adequate performance in scenarios with imbalanced data. Studies have shown that these models can still deliver satisfactory accuracy even under low-data conditions [15]. However, challenges such as domain-specific variations, noisy inputs, and imbalanced datasets remain prominent obstacles. These issues are particularly significant in practical applications and real-world scenarios, where inputs often lack uniform quality and training data is unevenly distributed [16]. Furthermore, there is a growing need to develop lightweight, optimized models that deliver satisfactory performance in resource-constrained environments, such as mobile devices or Real-Time Systems (RTSs). Advances such as compressed models and techniques like quantization and pruning Ganesh et al. [17] demonstrate efforts to reduce computational complexity and improve resource efficiency. At the same time, approaches such as multi-task learning and adjusting model weights based on token importance enable greater adaptability to imbalanced data [18]. Despite these advancements, designing models that can effectively balance accuracy, speed, and computational efficiency remains a fundamental challenge in the field of NLP. This issue is particularly evident in systems operating in environments with unstructured data [14].

In this paper, a novel architecture is proposed to enhance the performance of intent classification and slot filling using a transformer-based model equipped with Multi-Head Attention (MHA) Vaswani et al. [7], Root Mean Square (RMS) Normalization [12], and feedforward layers with Swish Gated Linear Unit (SwiGLU) activation [19]. The proposed model effectively captures long-range dependencies in user queries while preserving a robust representation of both intent and slot information. Furthermore, a two-stage adaptive learning approach is incorporated, wherein the model is initially trained on the full dataset and then fine-tuned on hard-to-classify samples to improve performance on complex, ambiguous cases. Experimental results demonstrate the effectiveness of the proposed approach in handling imbalanced data and low-frequency intents, thereby improving generalization across diverse conversational scenarios. The model outperforms

existing baselines in both accuracy and robustness, highlighting its potential for improving NLU in real-world applications.

The structure of this paper is as follows: in Section 2, the proposed method along with its details will be presented. In Section 3, the proposed method will be evaluated in two parts: Intent detection and slot filling. Finally, in the last section, the conclusion and summary of the findings will be discussed.

2 | Proposed Method

In this research, a novel transformer-based architecture for intent detection and slot filling is proposed, combining recent advancements to enhance computational efficiency and improve performance across a wide range of NLU tasks. This architecture is developed based on the standard transformer encoder and incorporates RMS normalization and SwiGLU activation in the feedforward sections. The primary goal of this design is to simplify the training dynamics of the model while maintaining and enhancing its representational capacity. The overall architecture of the proposed method is illustrated in *Fig. 1*. It begins with token and position embeddings, followed by MHA to compute relationships between tokens. RMS Normalization and residual connections improve stability and information flow. The feedforward layer with SwiGLU activation then processes the data. In the proposed method, multiple encoder layers refine the token feature vectors. Finally, a linear layer and a Softmax function are used to predict intent and slot labels. The details of each step are explained in the following sections.

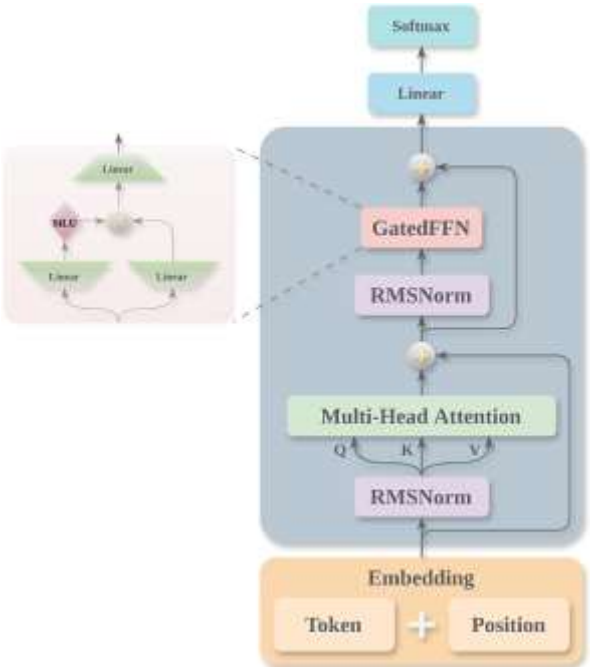

Fig. 1. Proposed transformer-based architecture for intent classification and slot filling.

Consists of three main components: embedding, a stack of encoder blocks, and a softmax layer. The overall structure of the model is identical for both tasks (intent classification and slot filling), with only a task-specific head added to the main model body. This approach improves computational efficiency.

The proposed model consists of three main components *Fig. 1*. Embedding: This section includes token and position embeddings, which convert textual inputs into numerical vectors. Stack of Encoder Blocks: This section comprises multiple transformer encoder layers utilizing MHA, RMS normalization, and feedforward layers with SwiGLU activation. These blocks extract long-range dependencies and refine token features. Softmax: This section is used to predict intent and slot labels. Each section is described in detail below.

2.1|Data Preprocessing

Input messages must first be converted into numerical vectors to be used as model inputs. For this purpose, all messages in the dataset are tokenized. All tokens are extracted from the messages, assigned a numerical value, and stored in a dictionary. This process allows us to replace each token with its corresponding numerical value during message encoding into the input vector. The special token CLS is appended to the inputs as a learnable vector, and the label generation process is applied to this vector by the model. Other special tokens are as follows:

I. PAD Token: This token is used to standardize the length of entity batches.

II. UNK Token: Since the tokens in the dataset are limited, and future messages may contain tokens not present in the dictionary, this token is used for such cases. During encoding, the numerical value corresponding to this token is placed in the input vector to ensure the model's efficiency and accuracy are not compromised when encountering unknown inputs.

III. SEP Token: This token is used to separate messages.

2.2|Architecture of the Proposed Method for Intent Detection and Slot Filling

Input tokens enter the Token Embedding section. In this section, each token from the encoded message vector is mapped to its corresponding embedding. Since the transformer model lacks an inherent understanding of the positional and sequential arrangement of tokens, the token embeddings must be enriched with positional information. This positional information enables the model to recognize each token based on its position, as the meaning of a token can vary depending on its context within the sequence.

$$X_0 = \text{TokenEmbedding}(\text{Tokens}) + \text{PositionalEmbedding}. \quad (1)$$

In the next step, the encoder receives the resulting vector as input. In this section, the model gains the ability to understand the semantic relationships between input tokens. The core of the transformer model is the transformer block. The power of transformers comes from stacking multiple blocks (in this approach, $N_{Encoder} = 4$), allowing the model to learn complex and abstract relationships between input tokens. This section consists of three main components:

2.2.1|Root mean square normalization

Instead of using traditional LayerNorm (LN), we apply RMS normalization before the Self-Attention (SA) mechanism. RMS stabilizes the training process by normalizing inputs based on the RMS of their values, resulting in lower computational overhead compared to LN. RMS scales the input without shifting it, which can accelerate convergence during training.

$$X_{norm} = \text{RMSNorm}(X_0), \quad (2)$$

$$\underline{a_i} = \frac{a_i \times w_i}{\text{RMS}(a)}, \text{where RMS}(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i{}^2}. \quad (3)$$

2.2.2|Multi-head attention

To extract relationships between tokens, we use the Multi-Head Self-Attention (MHSAM) mechanism. This component allows the model to simultaneously focus on different parts of the sequence. The input sequenceX_0 is transformed into Queries, Keys, and Values, which are used in the SA mechanism. The outputs of each attention head are concatenated and projected back to the original dimensions.

$$X_{attn} = \text{Attention}(X_{norm}) + X_0. \quad (4)$$

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_{model}}}\right)V, \text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_k)W^0. \quad (5)$$

$$\text{where head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right). \quad (6)$$

2.2.3 | Feedforward network with SwiGLU

Transformers have emerged as a dominant architecture in NLP, achieving state-of-the-art results across a wide range of tasks. However, their ability to model complex dependencies and capture nuanced patterns in data can be constrained by the limitations of standard feedforward networks, which are commonly used in most transformer-based models. These traditional feedforward layers often struggle to efficiently represent intricate relationships within the data, potentially hindering the model's overall performance.

$$X_0 = \text{Linear}\left(\text{SiLU}(X_{norm})\right) \times \text{Linear}(X_{norm}) + X_0. \quad (7)$$

To address this limitation, the introduction of Gated Linear Units (GLU) variants, such as SwiGLU, into the feedforward layers provides a novel and effective approach to enhance the model's representational capacity. GLU-based layers incorporate gating mechanisms that allow the model to selectively process information, enabling it to better capture complex dependencies and improve its ability to generalize across diverse datasets. By integrating SwiGLU, the model gains the ability to dynamically adjust the flow of information, leading to more efficient learning and improved performance. One of the key advantages of using SwiGLU is its ability to boost the model's efficiency without significantly increasing computational overhead. This characteristic makes it particularly suitable for large-scale applications where computational resources are a concern. Additionally, SwiGLU's design allows for smoother gradient flow during training, which can accelerate convergence and improve the stability of the learning process.

3 | Intent Classification

The proposed model is trained separately for both intent classification and slot filling tasks. However, the model's architecture remains consistent for both applications, with only task-specific output layers added at the end of the model during training. This approach allows for weight sharing and leveraging learned features in the shared layers, while the specific tasks of intent classification and slot filling are handled by dedicated output layers. In this section, the details and performance of the model in user intent classification are examined. This part of the model is responsible for identifying the user's goal or intent by analyzing input messages. The model's performance in this task directly impacts the quality of the system's responses, as accurately detecting the user's intent is the first critical step in providing relevant and useful answers.

3.1 | Two-Stage Training

In addition to the innovative architecture, a novel training approach has also been employed for the proposed model in this study. This method involves a two-stage training process, which enhances the model's performance in intent classification and slot filling. In the first stage of training, the model is trained on the initial dataset, establishing a baseline performance. Then, in the second stage of training, the model is retrained on the misclassified samples from the first stage. This approach allows the model to focus more on difficult examples, better capture complex dependencies between labels, and achieve higher generalizability when dealing with imbalanced data. The integration of this process with Focal Loss helps the model become more sensitive to low-frequency classes and improves its prediction accuracy across all classes in a balanced manner.

3.2 | Model Training Details

The proposed method was implemented using Python and the PyTorch framework. Model training was conducted on a system running the Ubuntu 22.04 operating system, equipped with an NVIDIA GeForce Ray

Tracing Texel eXtreme (RTX) 3050 Ti Mobile Graphics Processing Unit (GPU) with 4 GB of memory, ensuring efficient computation and accelerated deep learning processing. The model, initialized with the described hyperparameters, is ready for training. The goal is to minimize the error calculated by Focal Loss using the AdamW optimizer over the training dataset. The maximum number of tokens in a message (Max Positions) is set to 128 based on the dataset. After the mentioned preprocessing steps, the training data is fed into the model in batches of 8 (batch size = 8). The model's output and the true intent of the messages are compared using the mentioned loss function, and the model's error is calculated. After computing the error, the optimizer updates the model's parameters. The error and accuracy curves during the training process are shown in *Fig. 3*.

In the provided dataset, one of the main challenges is data imbalance. This issue causes the model to favor predicting the majority class, leading to reduced performance on rare classes. To address this problem, Focal Loss is used as the loss function for user intent detection. Focal Loss focuses on harder samples, creating a better balance between majority and minority classes and preventing the dominance of overrepresented classes. Additionally, the dataset contains difficult-to-classify samples, known as hard samples. In the proposed method, Focal Loss is employed to ensure that the model can effectively classify and learn from these hard samples. By assigning higher weights to hard samples during training, this loss function helps the model overcome challenges posed by imbalanced data and difficult-to-classify instances.

3.3 | Analysis and Evaluation of Results

In this section, we analyze and evaluate the results obtained from the proposed model, along with its strengths and weaknesses. Additionally, the details of the dataset used will be examined.

3.3.1 | Evaluation metrics

To evaluate the performance of the proposed model, Precision, Recall, and F1 Score metrics are used. These metrics are widely employed in NLP tasks, particularly in intent detection and slot filling.

Precision

This metric measures the ratio of correct predictions made by the model to the total predictions. High precision indicates that the model rarely makes incorrect predictions.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{8}$$

Recall

This metric measures the ratio of correct predictions to the total actual samples in the ground truth. High recall indicates that the model has a good ability to identify samples belonging to each class.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{9}$$

F1 Score

This metric is the harmonic mean of precision and recall, balancing the two. The F1 score is particularly useful when dealing with imbalanced datasets, as it offers a comprehensive evaluation of the model's overall performance.

$$\text{Precision} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{10}$$

Using these metrics together allows us to assess the model's performance from different perspectives. For instance, in imbalanced datasets, high precision might result from the dominance of majority classes, while

low recall indicates the model's weakness in identifying rare classes. The F1 score, by combining these two metrics, provides a clearer picture of the model's overall performance.

3.3.2 | Dataset

The dataset used in this study (*Fig. 2*) consists of textual data labeled for two main tasks: Intent detection and slot filling. This dataset exhibits a good level of diversity, including samples from both majority and minority classes. However, one of the main challenges of this dataset is data imbalance, which causes the model to favor predicting majority classes. To address this issue, the Focal Loss function was employed. This dataset consists of 2067 samples, each belonging to one of the 21 defined intents. The distribution of intents in the training dataset is illustrated in *Fig. 2*. In this study, the dataset is divided into training and evaluation sets, containing 1867 and 200 samples, respectively.

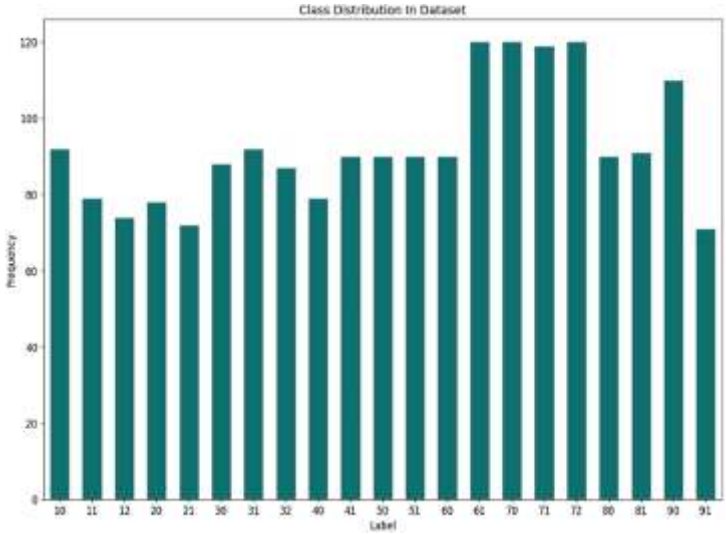

Fig. 2. Distribution of intents in the training dataset.

Such a distribution represents an unbalanced dataset, with some intents appearing significantly more often than others.

As shown in *Fig. 2*, classes such as 61, 70, 71, and 72 have a high frequency, while classes like 40, 12, and 91 are significantly less represented. This imbalance indicates that the model may learn the majority classes well but struggle with minority classes. Such an imbalance can lead to biased predictions, where the model predominantly favors majority classes, thereby reducing its overall effectiveness in real-world scenarios. The presence of several classes with a large number of samples suggests that these intents are common in user queries. The model is likely to perform well on these classes due to the abundance of training data. However, this can result in the model relying heavily on these majority classes and neglecting the rarer ones.

On the other hand, there are classes with a small number of samples in the dataset. These classes may not have sufficient data for the model to learn effectively. As a result, the model might completely overlook these classes or make errors when identifying samples belonging to them. The distribution of intents in the training dataset is shown in *Fig. 2*. The chart illustrates an imbalanced dataset, where certain classes have significantly more samples than others. This data imbalance can pose challenges for the model, as it may lead to a bias toward predicting majority classes, resulting in reduced performance for minority classes.

3.3.3 | Impact on evaluation metrics

The imbalanced distribution of classes can distort evaluation metrics such as Precision, leading to misleading results. For instance, high precision might be achieved by over-predicting majority classes, while the model performs poorly in identifying minority classes. This issue is particularly evident in datasets where some classes have significantly fewer samples. To avoid this bias and obtain a more accurate assessment of the model's performance, it is essential to use additional metrics such as Recall, Precision, and F1 Score. These metrics collectively evaluate the model's ability to correctly identify both majority and minority classes. Specifically,

the F1 score, by combining precision and recall, provides a more balanced view of the model's performance and is particularly useful for evaluating imbalanced datasets. Based on the results in Table 1, the proposed model has maintained a fair distribution across classes and avoided bias toward high-frequency classes. The average error rate of the model in these evaluations is calculated to be 1.82%, indicating its stable and accurate performance. This error rate is significantly lower compared to reference models, such as deep neural network-based models or other transformer-based methods.

Additionally, the Macro-F1 values show a negligible difference compared to Micro-F1, indicating the model's uniform performance across both low-frequency and high-frequency classes. The results of this table specifically demonstrate that the proposed model not only performs well in detecting high-frequency classes but is also successful in identifying low-frequency classes. These characteristics make the proposed model highly suitable for real-world applications, especially in scenarios where data is imbalanced. In the following sections, further details on the tuning of the model's parameters will be provided. *Table 1* shows training results using cross-validation. This table presents the training results of the proposed model using cross-validation. Each row in the table represents a Fold, for which metrics such as Recall, Precision, F1 Score, and Accuracy have been calculated in both Micro and Macro forms. The results obtained from each Fold indicate that the proposed model has an error rate of 1.82%, demonstrating its accurate and stable performance across all Folds.

Table 1. Cross-validation results demonstrating fairness toward minority classes.

Fold	Metric						
	Accuracy	Macro			Micro		
		F1	Recall	Precision	F1	Recall	Precision
1	90.50	90.50	90.95	91.02	90.50	90.50	90.50
2	90.50	90.50	90.95	90.47	90.50	90.50	90.50
3	94.00	94.00	94.28	94.49	94.00	94.00	94.00
4	95.00	95.00	95.23	95.51	95.00	95.00	95.00
5	93.00	93.00	93.33	93.92	93.00	93.00	93.00

92.59±1.82.

3.3.4 | Hyperparameters

The hyperparameters considered for training the model are presented in *Table 2*. These hyperparameters include key settings such as learning rate, batch size, number of encoder layers, number of attention heads, and other parameters related to the model's architecture and training process. The appropriate selection of these hyperparameters plays a crucial role in achieving optimal model performance, and in this study, their optimal values have been determined using trial-and-error methods. In this approach, different values for each parameter were tested, and the model's performance was evaluated based on metrics such as accuracy, F1 score, and error rate. This process was repeated iteratively until a combination of hyperparameters that delivered the best performance was identified. Based on the determined hyperparameters, the proposed model offers high efficiency while being lightweight and simple. The use of a limited number of layers (4 layers) and attention heads (4 heads) in the transformer architecture, along with optimal settings such as batch size (8), low learning rate (5e-5), and dropout mechanism (0.1), has enabled the model to effectively learn complex dependencies in the data without requiring heavy computational resources.

Additionally, the use of the AdamW optimizer, designed for better weight decay management, along with appropriate weight decay (Weight Decay = 0.01) and feedforward layer size (512), ensures the model's stable and accurate performance. These settings make the model particularly suitable for real-time applications and resource-constrained environments. The lightweight and efficient nature of this model makes it an ideal choice for implementation in scalable and practical systems, where a balance between accuracy and processing speed is crucial. In the following sections, the impact of these hyperparameters on various evaluation metrics such as F1, Recall, Precision, and Accuracy will be discussed.

Table 2. Hyperparameters considered for user intent detection.

Max positions	128	**Alpha, Gamma**	4, 1
Batch size	8	**Number of layers**	4
Dropout	0.1	**Number of heads**	4
$\mathbf{d_{model}}$	128	**Iterations**	1000

Learning rate	5e-5	**Optimizer**	AdamW
FFN size	512	**Weight decay**	0.01

The optimal values for these parameters were obtained through a trial-and-error approach and by training the model multiple times.

The Maya model, using 2,387 tokens and 1,382,528 parameters, has achieved an accuracy of over 95%. These results demonstrate the model's efficiency in extracting information and identifying intents, while its relatively low number of parameters reduces computational complexity and enables deployment in resource-constrained environments. These results are presented in *Table 3*.

Table 3. Values for the number of tokens, model parameters, and model accuracy.

Maya	Accuracy	Parameters	Tokens
	95.00	1,382,528	2,387

This table presents the values for the number of tokens, number of model parameters, and model accuracy. This information is used to evaluate the efficiency and performance of the model in user intent detection.

To assess the learning and training process of the model, *Figs. 3* and *4* illustrate the error and accuracy trends throughout the algorithm's learning process. In *Fig. 3*, the training error (train_loss) and testing error (test_loss) are shown over the course of training. Initially, both errors decrease rapidly, indicating that the model is effectively learning from the training data. This phase highlights the model's ability to quickly capture core patterns in the data. As training progresses, the decline in training error slows down and approaches a stable point, signifying that the model is converging. At this stage, further reduction in training error becomes difficult, confirming that the model has reached an optimal learning state and overfitting has been effectively prevented. Similarly, in *Fig. 4*, the training accuracy (train_acc) and testing accuracy (test_acc) are presented. At the beginning of training, both metrics increase rapidly, indicating that the model is effectively learning and identifying the key patterns in the data. Over time, the rate of increase in training accuracy slows down and approaches stability, signifying convergence. Such convergence suggests that the model has reached a point where further improvement in training accuracy is unlikely. As with training accuracy, the testing accuracy also increases and stabilizes, further demonstrating the model's ability to generalize to new, unseen data. The gradual improvement and subsequent stabilization of both training and testing accuracy confirm the model's stable and robust performance.

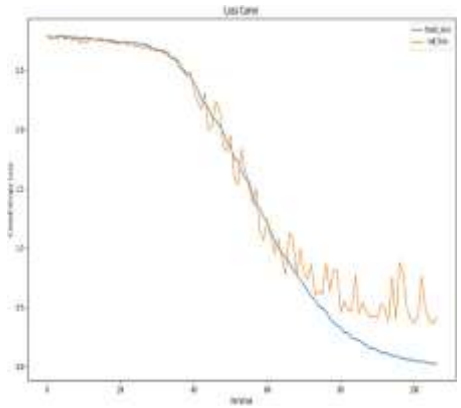

Fig. 3. Training and validation error trends during different learning stages of the proposed model.

This figure illustrates the changes in training error and validation error throughout the different learning stages of the proposed model. As observed, both training and validation errors gradually decrease and converge toward the end of the training process. This trend indicates the model's gradual improvement in performance and its effective learning from the data.

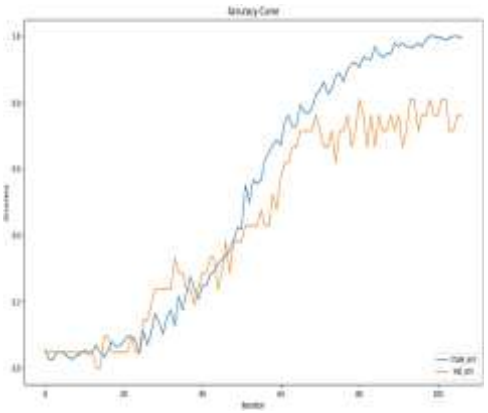

Fig. 4. Training and validation accuracy trends during different learning stages of the proposed model.

This figure illustrates the changes in training accuracy (train_acc) and validation accuracy (test_acc) throughout the different learning stages of the proposed model. As observed, both training and validation accuracies gradually increase and converge toward the end of the training process. This trend indicates the model's gradual improvement in performance and its effective learning from the data.

To thoroughly evaluate the effectiveness of the proposed model, two separate training phases were conducted. The first phase focused on training the model on the initial dataset, while the second phase aimed to improve its performance by retraining it on previously misclassified samples. The evaluation metrics used for assessing the model's performance include Accuracy, Precision, Recall, and F1 Score, both in Micro and Macro averaging modes. The following tables provide a comprehensive analysis of the model's performance across these two training phases. *Tables 4* and *5* present the evaluation metrics for the proposed model in two different training phases. In *Table 4*, the results of the first training phase, including Accuracy, Precision, Recall, and F1 Score in both Micro and Macro averaging modes, are shown. The obtained values for all metrics are nearly identical, ranging from 93.33% to 93.55%, indicating the model's high performance and stability in accurately identifying message intents. This consistency and high performance demonstrate that the model avoids bias toward majority classes and has a strong ability to learn from imbalanced data. *Table 4* presents the evaluation metrics of the proposed model in the second training phase, including Accuracy, Precision, Recall, and F1 Score in both Micro and Macro settings. The obtained results indicate that all metric values fall within the range of 97.62% to 98.20%, demonstrating a significant improvement compared to the first training phase. The close similarity between Micro and Macro values suggests that the model has Achieved Stable and optimal performance, maintaining a balanced classification across different classes. Such performance indicates that the model performs well not only in identifying frequent classes but also in detecting underrepresented ones. Furthermore, the high F1 Score highlights a proper balance between Precision and Recall, meaning the model effectively minimizes both false positives and false negatives. Such performance underscores the model's ability to accurately recognize message intents while maintaining strong and consistent performance even in imbalanced data scenarios. Overall, the results of this training phase confirm that the proposed model has successfully achieved high accuracy, well-balanced evaluation metrics, and stable performance, making it a reliable candidate for NLP systems in intent classification tasks. The results from both training phases collectively confirm that the proposed model demonstrates strong and stable performance, establishing it as a reliable and accurate solution for message intent detection, even in imbalanced data scenarios.

Table 4. Evaluation metrics for the proposed model in the first training phase.

Model	Training Phase	Metrics						
		Accuracy	Macro			Micro		
			F1	Recall	Precision	F1	Recall	Precision
Maya	1	93.33	93.33	93.33	93.55	93.33	93.33	93.33
Maya	2	97.62	97.62	97.62	98.20	97.62	97.62	97.62

This table presents the evaluation results of the message intent detection model for various metrics (accuracy, precision, recall, and F1 Score) in both Micro and Macro averaging modes. The obtained values for each metric are nearly identical, indicating the model's high performance and stability in accurately identifying message intents.

To evaluate the model's performance in intent detection and analyze classification errors in the two training phases, a confusion matrix was used. The confusion matrix is a useful tool for examining how the model classifies samples and identifying the types of errors it makes. It is particularly important in scenarios with imbalanced data, as it provides a more detailed view of how the model classifies samples into different classes and where it makes mistakes. *Figs. 5* and *6* display the confusion matrices corresponding to the proposed method in the first and second training phases. In *Fig. 5*, the results of the first training phase show several misclassifications. To reduce these errors and improve the model's performance, a second phase of training was conducted, where the model was retrained on the samples that had been misclassified previously. The results of this retraining phase, shown in *Fig. 6*, demonstrate a significant reduction in errors. Based on the confusion matrix in *Fig. 6*, which represents the second training phase, a noticeable reduction in errors compared to *Fig. 5* is observed, indicating a considerable improvement in the model's performance after the second training process. However, in some specific cases, the model still makes misclassifications when identifying user intents. In the test dataset, the model incorrectly classified 4 samples from class 90 into class 91 and 1 sample from class 10 into class 11. These errors suggest that classes 90 and 91, as well as classes 10 and 11, might share similar features, causing confusion for the model during the intent recognition process.

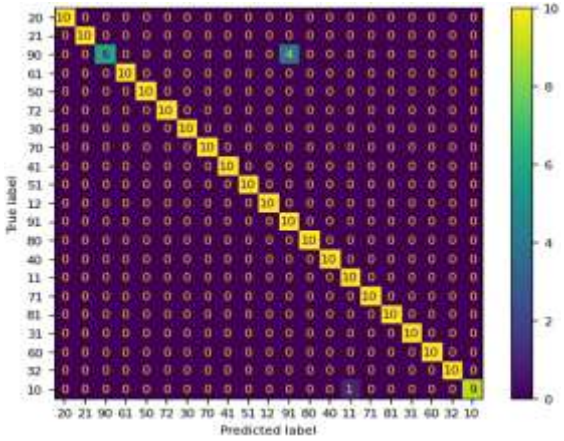

Fig. 5. Confusion matrix for user intent detection in the first training phase.

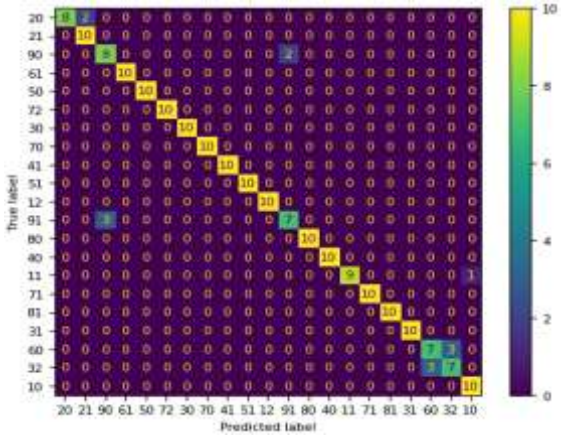

Fig. 6. Confusion matrix for user intent classification in the second training phase.

4 | Slot Filling

In this section, the CRF algorithm is utilized for model error calculation and decoding the sequence generated by the model. The optimizer, data preprocessing, and training process remain the same as in the intent classification section. In token labeling tasks, such as Named Entity Recognition (NER), Part-of-Speech (POS) tagging, and segmentation, the goal is to assign a label to each token in a sequence. Transformer-based models like BERT have demonstrated high effectiveness in this area due to their rich contextual embeddings. However, independent label prediction for each token might lead to ignoring dependencies between labels. CRFs are specifically designed to model these dependencies. A CRF is a probabilistic graphical model that defines the conditional probability distribution for a sequence of labels given the input data. Unlike simpler classifiers, CRFs consider dependencies between labels, making them well-suited for structured prediction tasks. The combination of transformers and CRFs leverages the advantages of both: 1) Transformers provide strong contextual embeddings, and 2) CRFs ensure structured label prediction. This hybrid approach delivers enhanced performance in many token-labeling tasks, particularly where label dependencies play a critical role.

To provide a detailed overview of the model configurations for the slot-filling task in messages, *Table 5* summarizes the key hyperparameters and architectural details used in the proposed model. These configurations are essential for optimizing the model's performance and ensuring its ability to accurately extract relevant slot information from messages. In this model, the Hidden Size is set to 128, allowing the model to extract more complex features from the data. The Max Positions is set to 128, enabling the model

to utilize limited positional information within messages. The model consists of 4 layers, reflecting its depth and capability in learning complex relationships. The Batch Size is set to 8, ensuring efficient training over the data. The model uses 4 attention heads, which dictate how attention is distributed across different parts of the input. A Dropout rate of 0.1 is applied to prevent overfitting. The number of iterations is set to 1000, determining the training duration of the model. Additional hyperparameters include a model dimension

(d_model) of 128, a learning rate of 5e-5, and a weight decay of 0.01. Finally, the FeedForward Network (FFN) size is 768, defining the dimensions of the model's sub-networks. These configurations enhance the model's ability to accurately and efficiently perform the slot-filling task in messages. To provide more details about the Maya model's features for the slot-filling task, *Table 6* presents some key characteristics of this model. In this model, the number of tokens is 2,387, while the number of parameters is 4,073,248, reflecting the model's complexity and capacity to learn patterns and relationships from the data. Finally, the model's accuracy is 84.28%, indicating the model's success rate in performing the designated prediction task.

Table 5. Hyperparameters considered in the user intent classification section. These hyperparameters were determined using a trial-and-error approach through multiple training iterations of the model.

Max positions	128	**Hidden size**	128
Batch size	8	**Number of layers**	4
Dropout	0.1	**Number of heads**	4
$\mathbf{d_{model}}$	256	**Iterations**	1000
Learning rate	5e-5	**Optimizer**	AdamW
FFN size	768	**Weight decay**	0.01

The optimal values for each parameter were selected to achieve the best model performance based on the Recall, Precision, F1, and Accuracy metrics.

Table 6. Values for the number of tokens, model parameters, and model accuracy.

Maya	Accuracy	Parameters	Tokens
	84.28	4,073,248	2,387

This table presents the values for the number of tokens, number of model parameters, and model accuracy. This information is used to evaluate the efficiency and performance of the model in user intent detection.

To assess the model's learning process and its ability to accurately identify intent, we analyze the trend of error reduction during training and evaluation. *Fig. 7* illustrates this progression by plotting the changes in Cross-Entropy Loss (CEL) across various stages of the model's training and evaluation. The horizontal axis represents the number of iterations, while the vertical axis shows the corresponding CEL values. This graph provides valuable insights into how well the model is converging, highlighting the reduction in error as the model learns from the training data. By examining the changes in loss, we can evaluate the efficiency of the training process and gauge the model's ability to generalize to new data.

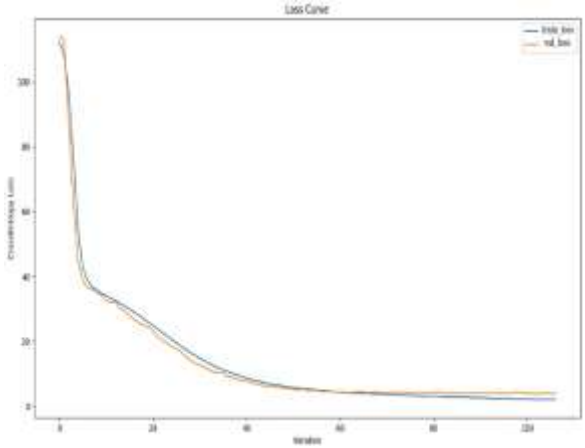

Fig. 7. Training and validation loss trend for slot filling.

To assess the model's effectiveness in performing the slot-filling task, *Table 7* presents the results. These results indicate that the Maya model performs well in filling message gaps. The Precision and Recall scores in the Macro metric are higher than in the Micro metric, suggesting that the model has performed uniformly across all classes, without showing significant bias toward any particular class. Additionally, the F1-score at the Macro level demonstrates a balanced trade-off between precision and recall, which is crucial for tasks

where both false positives and false negatives need to be minimized. This balanced performance further emphasizes the model's robustness in handling diverse and imbalanced data distributions.

Table 7. Training results for the slot filling task. This table presents the training results of the model for the slot filling task.

Model	Metrics						
	Accuracy	Macro			Micro		
		F1	Recall	Precision	F1	Recall	Precision
Maya	80.95	94.31	94.84	93.78	83.43	85.52	82.28

The evaluation metrics include F1 Score, Recall, and Precision, calculated in both Micro and Macro averaging modes.

To examine the impact of different architectures on the proposed method, *Table 8* presents various evaluation metrics for the Maya model, including Accuracy, Precision, Recall, and F1-score, measured at both Macro and Micro levels. This analysis helps evaluate how the model performs under different configurations and the effect of various layer structures and normalization techniques. The Maya (LN, MLP) model achieves the highest F1 Macro score (79.78%), indicating a relatively balanced performance across all classes. Such performance suggests that the model is effectively able to generalize and identify intents across various classes. In contrast, the Maya (RMS, Swish) model demonstrates lower Accuracy (44.29%) and Macro F1-score (61.14%), indicating weaker generalization. This reduced performance might be attributed to the added non-linearity introduced by the Swish activation function, which may not align optimally with this model configuration. The Maya (RMS, Swish, CRF) model shows the best Macro F1-score (62.89%) and Recall (63.30%), highlighting improvements in capturing minority class instances. The use of CRF seems to enhance the model's ability to handle class imbalance, making it more adept at identifying underrepresented classes or patterns. However, in terms of Micro-level metrics, the Maya LN, MLP model achieves the highest F1 (79.78%) and Precision (80.18%), reflecting stronger overall classification performance. Such performance indicates that the LN, MLP architecture excels in general classification tasks. The results suggest that incorporating CRF helps address class imbalance and improves Macro-level metrics, while the LN, MLP architecture maintains the best performance at the Micro

level. This analysis provides insight into the optimal architecture for various conditions and datasets, helping inform decisions on model configuration for specific tasks.

Table 8. Evaluation metrics for the proposed model with different architectures.

Model	Metrics						
	Accuracy	Macro			Micro		
		F1	Recall	Precision	F1	Recall	Precision
Maya (LayerNorm, MLP)	45.24	62.51	62.15	65.80	79.78	79.39	80.18
Maya (RMS, Swish)	44.29	61.14	61.87	63.06	77.59	78.33	76.87
Maya (RMS, Swish, CRF)	45.71	62.89	63.30	64.62	78.53	78.71	78.35

The metrics Recall, Precision, F1-score, and Accuracy have been calculated for each architecture.

To evaluate the model's performance in slot detection and to gain deeper insights into its prediction errors, an experiment was conducted on the test dataset. The aim of this experiment was to assess the accuracy of slot identification and to analyze the model's misclassifications for each class. This section presents the results of the misclassified slots by the model.

Fig. 8 illustrates the number of incorrectly predicted slots along with their corresponding class names. As shown, samples from class "O" have the highest number of errors. This class typically represents tokens that do not belong to any slot.

In the slot detection task on the test dataset, 98.71% of the slots were correctly identified. The incorrectly predicted slots and their counts are shown in *Fig. 8*.

Class "O"

Class "O" has the highest number of errors, indicating that the model struggles to distinguish this class from others. Such difficulty is likely due to the high frequency of this class or its similarity to other classes.

Other classes

Some other classes also have multiple errors, but these errors are significantly fewer compared to class "O". The distribution of errors across the remaining classes is heterogeneous, with some classes having very few errors, which may suggest that the model has learned to identify them better.

Poor performance on class "O"

The model performs poorly in distinguishing class "O" and requires further improvement in this area.

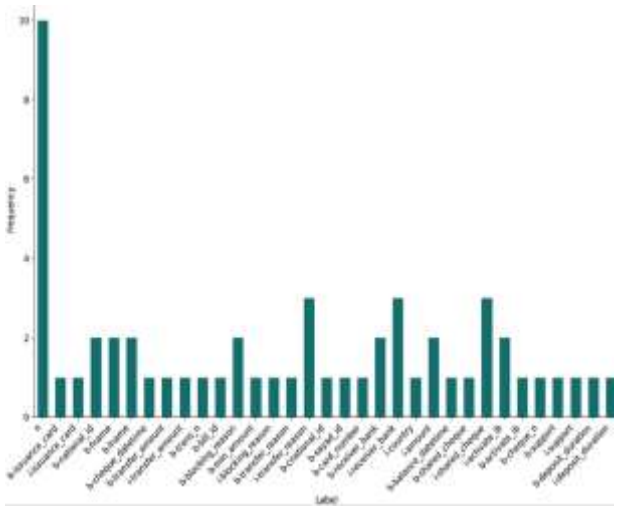

Fig. 8. Number of incorrectly predicted slots by the model, along with the corresponding class names.

As shown, the samples belonging to class "O" had the highest number of errors.

In order to assess the model's accuracy in slot detection and analyze the error rate in predictions, an experiment was conducted to examine the number of messages with incorrectly predicted slots. This experiment helps identify the model's weaknesses in correctly predicting slots and provides a clearer picture of its performance in detecting various slots. According to the results presented in *Fig. 9*, 19 messages had one slot incorrectly predicted, 8 messages had two slots incorrectly predicted, 4 messages had three slots incorrectly predicted, and 2 messages had four slots incorrectly predicted. This analysis helps understand the model's performance and limitations in the slot-filling task.

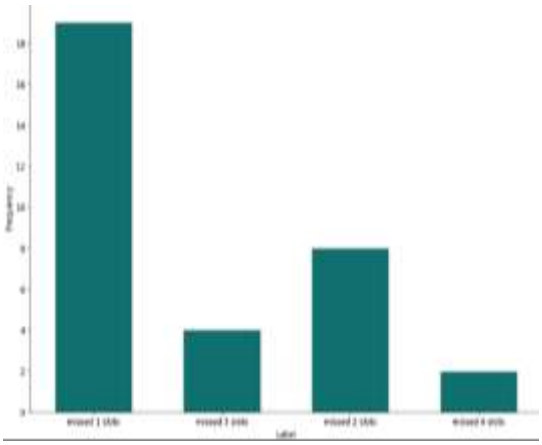

Fig. 9. This chart shows the number of slots that were incorrectly predicted by the model, along with the number of corresponding messages in the test dataset.

This chart helps better understand the distribution of errors and their relationship with the number of messages in each class.

5 | Conclusion

In this paper, a Transformer-based approach is presented along with advanced techniques such as Root Mean Square Normalization (RMSNorm), SwiGLU activation function, and CRF for intent classification and slot filling tasks in NLP. This architecture is designed to effectively extract long-term dependencies from the input data and mitigate biases present in imbalanced datasets. Additionally, the use of Focal Loss and a two-stage training approach on misclassified samples has improved the model's performance in identifying hard-to-predict instances and reduced errors. The results show that the proposed model significantly excels in intent classification and slot filling, efficiently handling imbalanced datasets and low-frequency classes. Furthermore, the model's optimal performance in intent classification has not only achieved high accuracy but also contributed to the reduction of false positives and false negatives. Given the model's efficiency and accuracy in resource-constrained environments with low computational requirements, this approach can serve as an effective framework for developing intelligent assistants and advanced conversational systems in the future. Several methods can be proposed to improve the model's performance in the future. One such method is data augmentation, where increasing the training samples for the "O" class could help the model better recognize this class. Additionally, data balancing techniques and applying various methods to reduce the impact of overrepresented classes in imbalanced datasets can assist the model in improving its performance on other classes. Furthermore, class weighting in the loss function for low-frequency classes or classes where the model performs poorly could help enhance the model's overall accuracy and effectiveness.

Authors' Contributions

All aspects of the research and manuscript preparation were carried out by the author. The author has read and approved the final version of the manuscript.

Funding

Not applicable.

Data Availability

All data supporting the reported findings in this research paper are provided within the manuscript.

Conflict of Interest

The author declares that they have no conflicts of interest.

Consent for Publication

The author confirms consent for the publication of this work.

Ethics Approval and Consent to Participate

This article does not contain any studies with human participants performed by the author.

References

[1] Tur, G., & De Mori, R. (2011). *Spoken language understanding: Systems for extracting semantic information from speech*. John Wiley & Sons. https://doi.org/10.1002/9781119992691

[2] Ray, S., & Craven, M. (2005). Supervised versus multiple instance learning: An empirical comparison. *Proceedings of the 22nd international conference on machine learning* (pp. 697–704). Association for Computing Machinery (ACM). https://doi.org/10.1145/1102351.1102439

[3] Pieraccini, R., & Huerta, J. (2005). Where do we go from here? Research and commercial spoken dialog systems. *Proceedings of the 6th SIGdial workshop on discourse and dialogue* (pp. 1-10). Special Interest Group on Discourse and Dialogue (SIGdial). https://doi.org/10.18653/v1/2005.sigdial-1.1

[4] Yao, K., Zweig, G., Hwang, M. Y., Shi, Y., & Yu, D. (2013). Recurrent neural networks for language understanding. *IEEE transactions on audio, speech, and language processing*, *21*(10), 2093–2103. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/338_Paper.pdf

[5] Goo, C. W., Gao, G., Hsu, Y. K., Huo, C. L., Chen, T. C., Hsu, K. W., & Chen, Y. N. (2018). Slot gated modeling for joint slot filling and intent prediction. *Proceedings of the 2018 conference of the north american chapter of the association for computational linguistics: Human language technologies, volume 2 (short papers)* (pp. 753–757). Association for Computational Linguistics. https://doi.org/10.18653/v1/N18-2118

[6] Zhang, X., & Wang, H. (2016). A joint model of intent determination and slot filling for spoken language understanding. *Proceedings of the twenty-fifth international joint conference on artificial intelligence* (pp. 2993–2999). AAAI Press. https://zxdcs.github.io/pdf/spoken_language_understanding.pdf

[7] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[8] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). Bert: Pre training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)* (pp. 4171-4186). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/N19-1423

[9] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I. (2018). *Improving language understanding by generative pretraining*. San Francisco, CA, USA. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[10] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). *Roberta: A robustly optimized bert pretraining approach*. https://doi.org/10.48550/arXiv.1907.11692

[11] Chen, Q., Zhuo, Z., & Wang, W. (2019). *Bert for joint intent classification and slot filling*. https://doi.org/10.48550/arXiv.1902.10909

[12] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., & Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, *32*. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf

[13] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text to text transformer. *Journal of machine learning research*, *21*(140), 1–67. https://www.jmlr.org/papers/volume21/20-074/20-074.pdf

[14] Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., ... & Stoyanov, V. (2020). Unsupervised cross lingual representation learning at scale. *Proceedings of the 58th annual meeting of the association for computational linguistics* (pp. 8440-8451). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/2020.acl-main.747

[15] Castellucci, G., Bellomaria, V., Favalli, A., & Romagnoli, R. (2019). *Multi lingual intent detection and slot filling in a joint bert based model*. https://doi.org/10.48550/arxiv.1907.02884

[16] Jiang, H., He, P., Chen, W., Liu, X., Gao, J., & Zhao, T. (2020). Smart: Robust and efficient fine tuning for pre trained natural language models through principled regularized optimization. *Proceedings of the 58th annual meeting of the association for computational linguistics* (pp. 2177-2190). Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.acl-main.197

[17] Ganesh, P., Chen, Y., Lou, X., Khan, M. A., Yang, Y., Sajjad, H., Nakov, P., Chen, D., & Winslett, M. (2021). Compressing larges cale transformer based models: A case study on bert. *Transactions of the association for computational linguistics*, *9*, 1061–1080. https://doi.org/10.1162/tacl_a_00413

[18] Gu, S., Zhang, J., Meng, F., Feng, Y., Xie, W., Zhou, J., & Yu, D. (2020). Token level adaptive training for neural machine translation. *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)* (pp. 1035–1046). Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-main.76

[19] Shazeer, N. (2020). *Glu variants improve transformer*. https://doi.org/10.48550/arXiv.2002.05202