Journal of Engineering and Technology for Industrial Applications

ITEGAM-JETIA

Manaus, v.11 n.51, p. 99-111. January/February., 2025.
DOI: https://doi.org/10.5935/jetia.v11i51.1423

ISSN ONLINE: 2447-0228

RESEARCH ARTICLE OPEN ACCESS

TRANSFORMER-BASED OPTIMIZATION FOR TEXT-TO-GLOSS IN LOW-RESOURCE NEURAL MACHINE TRANSLATION

Younes Ouargani[1] and Noussaima El Khattabi[2]

[1, 2] Laboratory of Conception and Systems (Electronics, Signals, and Informatics), Faculty of Science, Mohammed V University, Rabat, Morocco.

[1]http://orcid.org/0000-0002-0804-9218, [2]http://orcid.org/0009-0009-3390-275X

Email: younes_ouargani@um5.ac.ma, e.noussaima@um5r.ac.ma

ARTICLE INFO

Article History
Received: November 21, 2024
Revised: December 20, 2024
Accepted: January 15, 2025
Published: January 30, 2025

Keywords:
Real-Time Signal Processing,
Auditory Impairment,
Neural Machine Translation,
Optimization,
Sign Language

ABSTRACT

Sign Language is the primary means of communication for the Deaf and Hard of Hearing community. These gesture-based languages combine hand signs with face and body gestures for effective communication. However, despite the recent advancements in Signal Processing and Neural Machine Translation, more studies overlook speech-to-sign language translation in favor of sign language recognition and sign language to text translation. This study addresses this critical research gap by presenting a novel transformer-based Neural Machine Translation model specifically tailored for real-time text-to-GLOSS translation. First, we conduct trials to determine the best optimizer for our task. The trials involve optimizing a minimal model, and our complex model with different optimizers; The findings from these trials show that both Adaptive Gradient (AdaGrad) and Adaptive Momentum (Adam) offer significantly better performance than Stochastic Gradient Descent (SGD) and Adaptive Delta (AdaDelta) in the minimal model scenario, however, Adam offers significantly better performance in the complex model optimization task. To optimize our transformer-based model and obtain the optimal hyper-parameter set, we propose a consecutive hyper-parameter exploration technique. With a 55.18 Recall-Oriented Understudy for Gisting Evaluation (ROUGE) score, and a 63.6 BiLingual Evaluation Understudy 1 (BLEU1) score, our proposed model not only outperforms state-of-the-art models on the Phoenix14T dataset but also outperforms some of the best alternative architectures, specifically Convolutional Neural Network (CNN), Long Short Term Memory (LSTM), and Gated Recurrent Unit (GRU). Additionally, we benchmark our model with real-time inference tests on both CPU and GPU, providing insights into its practical efficiency and deployment feasibility.

I. INTRODUCTION

Sign languages, as visual-gestural forms of communication, are integral components of the linguistic landscape for the Deaf and Hard of Hearing (DHH) community. Unlike spoken languages, sign languages rely on visual and gestural elements, incorporating manual signs, body movements, and facial expressions to convey meaning. This unique modality enables individuals within the DHH community to express themselves with depth and nuance, offering a rich and diverse means of communication. The significance of visual-gestural languages becomes particularly evident when considering the limitations of traditional spoken languages in meeting the communication needs of the DHH community. Spoken languages heavily rely on auditory cues, making them less accessible for those with hearing impairments. In contrast, sign languages provide an inclusive and versatile medium that allows individuals to communicate effectively without dependence on auditory stimuli. Visual-gestural languages play a crucial role in facilitating social interactions, education, and professional engagement within the DHH community. The use of manual signs allows for the expression of abstract concepts, emotions, and complex ideas. Additionally, facial expressions and body language contribute to the linguistic richness of sign languages, enhancing the overall communicative experience. However, the broader societal landscape predominantly relies on spoken languages. This linguistic dissonance results in a noticeable

communication gap that Speech-to-Sign Language translation tools endeavor to address. However, for truly seamless interaction, real-time translation is crucial. Consider a classroom environment where a lecture is being translated, a delay can disrupt the flow of information and hinder understanding. The imperative development and implementation of these tools, particularly those with real-time capabilities, play a crucial role in facilitating effective and inclusive communication between individuals who use spoken languages and those who depend on visual-gestural languages, as these tools play a pivotal role in harmonizing interactions, promoting accessibility across linguistic and cultural boundaries, ensuring equal access to information, opportunities, and social interactions.

The evolution of machine translation has been a dynamic journey, progressing through various phases of development. It began with the early attempts of rule-based systems, as presented in [1], which explored the collaborative role of human translators and machines, emphasizing the synergy required for effective language translation in the early phases. These rule-based systems, however, faced challenges in capturing the complexities and nuances of language due to their reliance on predetermined linguistic rules. The subsequent transition to statistical methods marked a pivotal moment in machine translation's evolution. Brown et al.'s groundbreaking work [2] introduced probabilistic models that significantly enhanced translation accuracy by addressing linguistic variations. This departure from rigid rules allowed the model to learn patterns and relationships from data, enabling more nuanced and context-aware translations. The paradigm shift continued with the advent of neural networks. Ref [3] presented a transformative neural machine translation model with attention mechanisms. Unlike traditional models, this approach allowed the model to selectively focus on distinct sections of the input sequence amid translation. This attention mechanism proved crucial in handling long sentences and capturing contextual information, leading to substantial improvements in translation quality. Building upon this, [4] introduced the transformer architecture, which further refined the attention mechanism. The transformer architecture replaced recurrent layers with self-attention mechanisms, enabling the model to consider dependencies across the entire input sequence simultaneously. This innovation significantly improved the efficiency of training and the model's capacity to represent long-range dependencies, setting new standards in machine translation performance.

Sign language translation has followed a parallel evolution, incorporating diverse approaches to bridge the communication gap between spoken languages and visual-gestural languages. Early contributions include the work of According to [5], who presented a rule-based system for speech-to-sign language translation. Their focus on National Identification Document (NID) and Passport-related content demonstrates the practical application of translation systems. Transitioning to an alternative approach, the Arabic context highlights a significant achievement. An interdisciplinary team, collaborating closely with deaf native signers and an Arabic Sign Language (ArSL) interpreter, developed an example-based machine translation (EBMT) system [6]. This system adeptly translates Arabic text into ArSL, aligning with the unique linguistic nuances and cultural context of the Arabic deaf and hearing-impaired community. The choice of EBMT ensures adaptability to the intricate grammar, structure, and idioms inherent in ArSL.

In contrast to traditional rule-based approaches, recent advancements delve into the integration of Neural Machine Translation (NMT), showcasing the evolving landscape of sign language translation. According to [7] present an innovative synthesis by seamlessly integrating NMT and Generative Adversarial Networks to produce sign language video sequences from spoken language sentences. This approach not only showcases the capabilities of text-to-gloss translation but also underscores the potential for video generation in sign language translation offering a fresh perspective independent of avatars and motion-capture-based methods. In a different vein, [8] focus on the bidirectional translation of sign language, proposing a deep learning approach based on GRU and Long Short-Term Memory (LSTM) models with attention mechanisms. Their work stands out by demonstrating superior performance on ASLG-PC12 and Phoenix-14T corpora, particularly with the GRU model using Bahdanau attention [3]. This highlights the effectiveness of their approach in capturing the linguistic structure and context of natural sign language sentences. For Sign Language Production (SLP), Saunders et al. [9] provide a progressive Transformers architecture that performs end-to-end translation of spoken language sentences into continuous 3D sign pose sequences. Their method addresses the need for architectures more appropriate for continuous sign sequence generation by applying a counter-decoding methodology. By providing benchmarks on the complex PHOENIX14T dataset and establishing a baseline for subsequent studies, Saunders et al. emphasize the importance of continuous sequence synthesis and lay the groundwork for further exploration in the field.

These modern translation methods showcase a notable advancement over earlier techniques. However, several studies shed light on the critical role of hyper-parameter tuning in improving Transformer performance in low-resource neural machine translation (NMT) scenarios. In [10] Araabi and Monz run several experiments on subsets of the IWSLT14 training corpus, they highlight the influence of hyper-parameters on the performance of Transformer models under low-resource scenarios. This study underscores the importance of proper configuration, showing that optimizing Transformer hyper-parameters can lead to an improvement of up to 7.3 BLEU points in translation quality compared to using default settings. Additionally, [11] focus on deep Transformer optimization for translation tasks in low-resource conditions, specifically for Chinese-Thai machine translation. Their exploration of various experiment settings, including the embedding size, dropout probability, and number of BPE merge operations, highlights the significance of choosing optimal configurations, even when dealing with low-resource scenarios. This research reinforces the notion that hyper-parameter optimization is critical to enhancing the performance of Transformer models across different language pairs and data conditions.

Expanding on the existing body of research, and building on our previous work [12], our study investigates the intricacies of optimizing transformers for the real-time text-to-GLOSS translation task, marking the first comprehensive study in this domain. The exploration is initiated by crafting a minimal model for thorough optimizer screening, specifically Adaptive Delta (AdaDelta) [13], Stochastic Gradient Descent (SGD) [14], and Adaptive Moment Estimation (Adam) [15], Adaptive Gradient (AdaGrad) [16]. Subsequently, harnessing the potential of the best optimizers, we employ an innovative hyper-parameter exploration technique tailored for Transformers. This methodology enables us to discern the optimal architecture for sign language translation, thereby making a significant contribution to the field of NMT for sign language processing.

Within this scope, the key contributions of this study are:

- We propose a novel hyper-parameter exploration technique for Transformer-based architectures on low-resource tasks, and use it to create a real-time text-to-GLOSS sequence-to-sequence translation model.
- We investigate the performance of AdaDelta, SGD, AdaGrad, and Adam optimizers on low-resource sequence-to-sequence tasks and evaluate their performance on Transformer models.
- We evaluate the proposed model's performance on the PHOENIX-14T corpus, demonstrating superior performance using the Bi-Lingual Evaluation Understudy (BLEU), and the Recall-Oriented Understudy for Gisting Evaluation (ROUGE) metrics.
- We compare our model's performance with other sequence-to-sequence architectures like GRU, CNN, and LSTM.

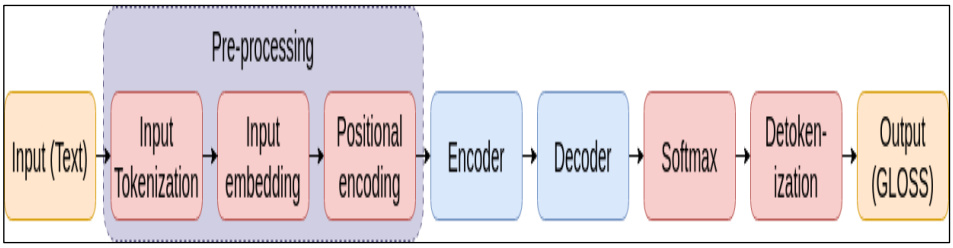

Figure 1: Proposed pipeline for Text-to-GLOSS Neural Translation
Source: Authors, (2025).

I.1. PROPOSED METHODOLOGY

In this section, we will present the resources necessary to understand our methodology approach and provide an overview of each of these elements. First, we will introduce our pipeline and describe each of the steps used to process the input text and generate the GLOSS output, then we will address the transformer architecture as it is a critical component of our architecture, then we will present the used optimizers in detail as they are a key element in producing a high fidelity translation model, then we are going to discuss the used performance metrics as it is extremely challenging to compare each model and optimizer's performance without an adequate performance metric.

II. REAL-TIME TEXT-TO-SIGN LANGUAGE GLOSS TRANSLATION PIPELINE USING TRANSFORMERS.

Our real-time text-to-sign language GLOSS translation pipeline encompasses a series of essential steps to enable a seamless conversion of textual input into a sign language GLOSS representation. As can be seen in Figure 1, the process begins with input tokenization, which divides the source text into discrete units for processing. Subsequently, input embedding encodes these tokens into vector representations, while positional encoding introduces spatial information to maintain word order.

The heart of the pipeline features the transformer architecture, consisting of an encoder that captures context and dependencies within the input text and a decoder that generates the corresponding sign language gloss. The SoftMax layer assigns probabilities to different gloss elements. Finally, detokenization reconstructs the output into a coherent GLOSS that represents the signed expression of the original text.

Each of the pipeline steps involves the following:

- **Input Tokenization:** The input text is divided into smaller units called tokens, which can be individual words or subwords. Tokenization is essential for representing text data in a format that the Transformer model can process.
- **Input Embedding:** Each input token is mapped to a high-dimensional vector representation called an embedding. The embedding layer helps the model to capture the semantic meaning of the tokens and their relationships within the input sequence.
- **Positional Encoding:** Since the Transformer architecture does not have built-in mechanisms to handle the sequential order of the input tokens, Positional encoding serves to provide information about the token's position in the sequence. By appending positional encoding to the input embeddings, the model is better able to comprehend the input data's sequential structure.
- **Encoder:** A series of transformer layers is used to process the input token embeddings with positional encodings. The input sequence is processed by the encoder, generating a sequence of continuous representations that capture the learned information for each token.
- **Decoder:** A set of transformer layers is applied to the encoder's output. The decoder's role is to generate the output sequence, which in this case is the gloss.

Softmax: The output of the decoder is subjected to an activation function to obtain a probability distribution over the possible glosses. The equation of the softmax function is:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad (1)$$

with z_i the i^{th} element of the input vector, and K the number of elements in the input vector.

Detokenization: Convert the predicted gloss back into text form. This is the reverse process of tokenization.

II.1 TRANSFORMER-BASED MODEL ARCHITECTURE.

Our text-to-GLOSS translation model, based on transformers, is inspired by the well-established encoder-decoder architecture commonly found in neural models for sequence transduction [17],[18]. This structural choice is vital for preserving the task's sequential aspect, allowing the production of GLOSS outputs that are both coherent and contextually accurate.

Incorporating a sophisticated attention mechanism, the two primary components of our model's architecture are an encoder and a decoder. The encoder maps a sequence of input symbol representations $(x_1, \ldots, x_n)$ to an output sequence $(z_1, \ldots, z_n)$, while the output sequence $(y_1, \ldots, y_n)$ is generated by the decoder in an autoregressive manner. This sequential generation preserves the integral temporal dependencies within the translation process,

ensuring that each output element is generated based on the symbols produced previously.

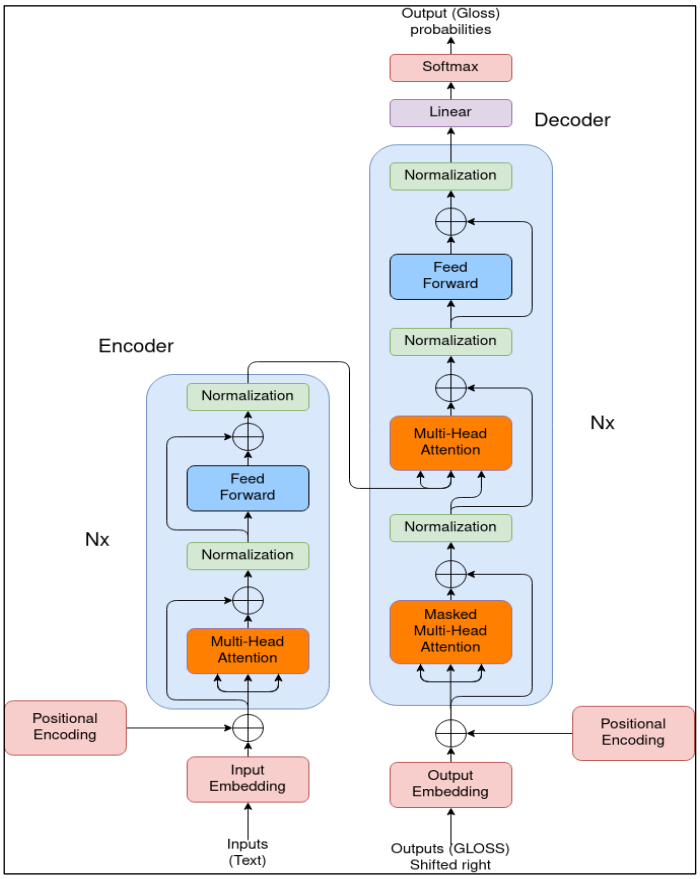

Figure 2**:** Proposed transformer-based model architecture for text-to-GLOSS translation.
Source: Authors, (2025).

Figure 2 shows the Transformer architecture, which is well-known for its effectiveness in processing sequential data via a combination of layered self-attention and dense layers, providing a visual representation of both the encoder and decoder components. Consisting of *N* identical layers, each comprising two sub-layers, the encoder integrates a multi-head self-attention mechanism into its first sub-layer. The second sub-layer employs a position-wise fully connected feed-forward network. Each sub-layer is surrounded by a residual connection [19], and to ensure smooth information flow, layer normalization [20] is applied after the residual connection. Notably, an output dimension *d* is maintained across all sub-layers, aligning with the dimension of the embedding layer. Similarly, the decoder is structured with a stack of *N* identical layers, augmented by a multi-head attention sub-layer designed to handle the output of the encoder stack. This is followed by a residual connection and a normalization layer. Thus, the output of each sub-layer in the model can be expressed as:

$$LayerNorm(x) = \left(x + Sublayer(x)\right) \quad (2)$$

where *Sublayer* represents the function applied by the sub-layer and *LayerNorm* denotes layer normalization.

To maintain the autoregressive nature of the decoder, information flow from subsequent positions is prevented by masking the self-attention sub-layer. To achieve this, the output embeddings are shifted one position and masked to ensure only known outputs from preceding positions are used to predict the output at position *i*.

The attention mechanism has become an integral part of sequence-to-sequence and transduction models. Attention allows modeling dependencies regardless of their distance in the input and output sequences. In our model, we use a combination of scaled dot-product and multi-head attention. The scaled dot-product attention computes the dot products of the query and keys, divides each by the square root of the dimension of the keys, and applies a SoftMax function to obtain the weights on the values. It is computed as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (3)$$

where the values, keys, and queries are packed together into the V, K, and Q matrices accordingly. And d_k is the dimension of the input keys (and queries).

The multi-head attention mechanism on the other hand uses several parallel attention layers, with each attention layer (head) having its own set of queries, keys, and values. This is expressed by the equation:

$$MultiHead(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O \quad (4)$$

with $\text{head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$. Where Q, K, and V are the query, key, and value matrices respectively, h is the number of attention heads, W_i^Q, W_i^K, and W_i^V projections are the parameter matrices for the i-th attention head, and W^O is the output projection matrix.

II.2 MODEL OPTIMIZATION

Optimizers are essential for training deep learning models, as they determine how the model's parameters are updated based on the gradients of the loss function. A good optimizer can significantly improve the convergence speed and final performance of the model. In the context of transformers, which are complex and computationally intensive models, choosing the right optimizer is essential for efficient training and inference.

- **Stochastic Gradient Descent(SGD)** [14]: is a widely used optimizer in deep learning. It updates the model's parameters by taking small steps in the direction of the loss function's negative gradient. The learning rate, which determines the step size, is a critical hyper-parameter that requires precise adjustment. The SGD algorithm utilizes the following equation to update the model's parameters:

$$\theta_{t+1} = \theta_t - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta L(f(x_i; \theta_t), y_i) \quad (5)$$

where θ_t represents the model parameters at iteration t, η the learning rate, which indicates the step size in the parameter update, $\nabla_\theta L(f(x_i; \theta_t), y_i)$ the gradient of the loss function with respect to the model parameters at iteration t (evaluated on a batch of training examples (x_i, y_i))

- **Adaptive Gradient (AdaGrad)** [16]: is an optimizer that adapts the learning rate for each parameter based on its past gradients. It performs smaller updates for parameters that are frequently updated and larger updates for infrequently updated parameters. The AdaGrad algorithm updates the parameters according to the following equations:
  - The first step is to compute the gradient g_t of the loss function with respect to the parameters θ.

- Update the squared sum of the gradients: $G_t = G_{t-1} + g_t^2$
- Update the parameters: $\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{G_t+\epsilon}} \cdot g_t$

where G_t is the sum of the square gradients up to the iteration t, η is the learning rate that controls the step size in the parameter update, and ϵ is a small constant added for numerical stability.

- **Adaptive Delta (AdaDelta)** [13]**:** is a stochastic gradient descent method that extends AdaGrad and seeks to address its limitations. Like AdaGrad, AdaDelta maintains a per-parameter learning rate, but it also introduces a decay term that helps to prevent the learning rate from becoming too small. This decay term allows AdaDelta to adapt to changing data and model parameters. The AdaDelta algorithm functions as follows: First the accumulation variables $E[g^2)$ and $E[\Delta x^2)$ are initialized to zero; then to update each parameter x at time t:

- Compute the gradient g_t
- perform a gradient accumulations step: $E[g^2)_t = \rho E[g^2)_{t-1} + (1-\rho)g_t^2$
- Computes the parameter update: $\Delta x_t = -\frac{RMS[\Delta x)_{t-1}}{RMS[g)_t} g_t$
- Accumulates updates: $E[\Delta x^2)_t = \rho E[\Delta x^2)_{t-1} + (1-\rho)\Delta x_t^2$
- Apply the updates: $x_{t+1} = x_t + \Delta x_t$

where x_t is the current value of the model's parameters, ρ is the decay rate, g_t is the gradient of the loss function in relation to the model parameters at time step t, Δx_t is the update to the parameter, and $RMS[\cdot)$ is the root mean square of the values.

- **Adaptive Moment Estimation (Adam)** [15]: combines the advantages of two popular optimizers: AdaGrad, which is advantageous in sparse gradient situations, and RMSProp, which excels in online and non-stationary settings. It uses the first and second moments of the gradients to adapt the learning rate for each parameter. The update rule for Adam is given by:

- Compute the gradient g_t.
- Update the first momentum estimate: $m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t$
- Update the second momentum estimate: $v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2$
- Correct the bias of the first moment estimate: $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$
- Correct the bias of the second moment estimate: $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$
- Update the parameters: $\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t+\epsilon}} \hat{m}_t$

where m_t and v_t are the first and second moments of the gradients, respectively, $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected estimates of the moments, θ_t is the current value of the model's parameters, g_t is the gradient of the loss function with respect to the parameters at time step t, α is the learning rate, β_1 and β_2 are the exponential decay rates for the moment estimates and ϵ is a small constant added for numerical stability.

II.3 MODEL EVALUATION

In the evaluation of the text-to-gloss transformer pipeline, we employed several metrics to assess the performance of the generated glosses. Each metric serves a specific purpose and provides valuable insights into different aspects of the generated text. The following metrics were used:

Perplexity: is a commonly used metric to measure the quality of language models. It measures the accuracy with which a model predicts a sample of text. More precisely, it measures how well a model predicts the following word in a series based on the preceding word sequence. Models demonstrating higher predictive capabilities over a text sample are characterized by a lower perplexity score.

Bi-Lingual Evaluation Understudy(BLEU) [21]: is a metric that measures the similarity between a generated text and one or more reference texts. It is often used in machine translation tasks but can also be applied to other text generation tasks such as text summarization, and image caption generation. BLEU scores range from 0 to 1 with a higher score indicating a better translation quality. The BLEU score also incorporates a brevity penalty to penalize translations that are shorter than the reference text. The driving factor behind the use of this penalty is that shorter translations in addition to being easier to generate are more likely to have a higher n-gram precision.

Recall Oriented Understudy for Gisting Evaluation (ROUGE) [22]: is a set of metrics used for evaluating automatic summarization of texts as well as machine translations. It evaluates the quality of summaries or translations by comparing them to a set of reference summaries. It measures the overlap between the generated summary and the reference summaries in terms of n-gram matches and word sequences. ROUGE scores range from 0 to 1, with higher scores indicating better performance. Out of the ROUGE score variations, we specifically use the ROUGE-L which uses the Longest Common Subsequence.

II.4 REAL-TIME INFERENCE

For seamless communication in real-time scenarios, achieving fast and efficient translation is crucial. We leverage CTranslate2 [23], a custom C++ Transformer-specific inference engine, to enable real-time deployment of our text-to-gloss translation model. CTranslate2 offers a significant advantage by having no runtime dependencies on TensorFlow or PyTorch. This eliminates potential compatibility issues and streamlines deployment. Additionally, CTranslate2 demonstrates optimized inference capabilities, including CPU and GPU support, leading to up to 4 times faster translation speeds compared to PyTorch [23]. This focus on real-time performance allows our model to be used in applications like live captioning and educational settings, providing greater inclusivity for the Deaf and hard-of-hearing community.

III. EXPERIMENTAL RESULTS

In this section, we present the results of our experiments, which demonstrate the effectiveness of the transformer in text-to-gloss translation tasks and provide valuable insights into the performance of this model in this specific task. The results are organized as follows: In the first subsection we introduce the experimental setup used to run our experiments, then the dataset subsection will provide more details about the dataset used for training and benchmarking our models, and finally, a results subsection that presents our consecutive hyper-parameter exploration, followed by a model optimization section, before finally presenting the performance results we share the obtained optimal parameters of our final model.

III.1 EXPERIMENTAL SETUP

Our text-to-GLOSS transformer model was built using the open source OpenNMT toolkit [24] with a pytorch backend [25]. The experiments were performed on a PC with an Intel Core I5

Central Processing Unit, an Nvidia RTX 3060 Graphics Processing Unit, 16GB of Random-Access Memory, and an Ubuntu Operating System.

Table 1: Text-to-Gloss Examples from the PHOENIX14T Dataset.

Text	GLOSS
AM SAMSTAG IST ES WIEDER UNBESTÄNDIG	SAMSTAG WECHSELHAFT
AUCH AM SAMSTAG TEILWEISE FREUNDLICH .	SAMSTAG AUCH FREUNDLICH
SONST SCHEINT VERBREITET DIE SONNE .	SONST REGION SONNE
IM SÜEDOSTEN REGNET ES TEILWEISE LÄNGER .	SUEDOST DURCH REGEN

Source: Authors, (2025).

Consistency in scoring methodologies is essential for establishing reliable benchmarks and facilitating meaningful comparisons between different models and research studies. The use of standardized scoring scripts helps to mitigate discrepancies in evaluation and ensures that the reported results are directly comparable. For this purpose, we specifically selected the Moses multi-bleu-detok.perl script for BLEU scoring, as it's used extensively used to report BLEU scores in research. To ensure a uniform ROUGE scoring and an accurate ROUGE comparison with other studies, we used HuggingFace's rouge scoring script which is a wrapper of Google Research's native Python implementation of ROUGE scoring.

III.2 DATASET

The PHOENIX-14T parallel text-to-GLOSS corpus [26] was employed to assess the performance of our proposed model. It is developed at RWTH Aachen University in Germany by the Human Language Technology & Pattern Recognition Group as part of the RWTH-PHOENIX-Weather 2014 corpus [27].

Table 2: PHOENIX14T Dataset Distribution.

	GLOSS			TEXT		
	Train	**Dev**	**Test**	**Train**	**Dev**	**Test**
Sentences	7 096	519	642	7 096	519	642
Words	67 781	3 745	4 257	99 081	6 820	7 816
Vocabulary	1 066	393	411	2 887	951	1 001

Source: Authors, (2025).

The dataset encompasses German sign language interpretation in the form of high-quality video recordings sourced from daily weather forecasts and news from 2009 to 2011. Additionally, the original German speech has been transcribed using a combination of speech recognition and manual cleaning. Manual GLOSS notation for German Sign Language (DGS) is available for 386 editions of weather forecasts. Some examples of the parallel text GLOSS dataset are provided in Table 1, and detailed statistics of the dataset's sentence, word, and vocabulary count are provided in Table 2.

The PHOENIX14T dataset proves to be a valuable asset for our research for several compelling reasons. Firstly, it is a non-synthetic dataset, offering accurate interpretations in German Sign Language (DGS) from professional interpreters. This authenticity is important for developing a high-performing system that can deliver accurate translations in uncontrolled environments. Additionally, the dataset is widely utilized in the sign language recognition and translation field, indicating its relevance and reliability for training text-to-GLOSS translation systems. Furthermore, its adoption facilitates the establishment of a standardized evaluation for our proposed architecture through a comparative analysis of our findings with state-of-the-art models. Despite its relatively small size, the dataset is comprehensive, offering a diverse range of linguistic and visual data for robust model training and evaluation. This further solidifies its suitability for the development of the text-to-GLOSS neural translation system.

III.3 RESULTS

III.3.1 Hyper-parameter Exploration

In this subsection, we introduce a novel transformer-based text-to-GLOSS translation architecture, considering the challenges posed by the limited resource conditions of the task. Thanks to their ability to capture contextual information and model long-range dependencies, Transformers have demonstrated remarkable success in various natural language processing tasks, especially in NMT. However, while having considerable accomplishments in NMT, their optimal utilization in the text-to-GLOSS translation endeavor remains to be comprehensively explored. Achieving this potential requires thorough hyper-parameter optimization, a vital operation for achieving optimal performance in low-resource scenarios.

Identifying the optimal Transformer architecture using grid search for a comprehensive exploration of hyper-parameters can be prohibitively expensive. Consequently, researchers resort to one of two techniques: a random hyper-parameter exploration [28] or an individual hyper-parameter grid search. While random search may yield superior hyper-parameter combinations, it typically incurs a greater time expense for a comprehensive exploration. Alternatively, grid search, for a single hyper-parameter at a time, only identifies the best value for the current hyper-parameter set, which is unaltered even after adjusting other hyper-parameters. This prompted us to adopt a consecutive hyper-parameter exploration approach. This method remedies the aforementioned downfalls by consecutively refining the model's hyper-parameters and not only relying on one round of optimization. The hyper-parameter range outlined in Table 3 is used to carry out this exploration.

Table 3: Explored hyper-parameter space.

Hyper-parameter	Values
Warmup steps	100 200 300 400 500 600
Batch-size	256 512 1024 2048 4096
Attention heads	1 2 4 8
Number of layers	1 2 3 4 5 6 7
Embedding dimension	32 64 128
Feed-forward dimension	128 256 512
Dropout	0.1 0.2 0.3 0.4 0.5
Label smoothing	0.1 0.2 0.3 0.4 0.5 0.6

Source: Authors, (2025).

During the iterative process of consecutive hyper-parameter exploration, a methodological approach is applied to refine the transformer-based architecture. The methodology involves the careful selection of an initial set of hyper-parameters, followed by sequential optimization of each parameter. Each hyper-parameter is individually addressed while keeping the others constant, and the model's performance on the selected dataset is assessed. The results of the first optimization cycle are used to modify the hyper-parameter values. Iteratively fine-tuning the model for each hyper-parameter separately until there are no more gains in improvement

to the model's output. By using this strategy of exploring hyper-parameters sequentially, it is possible to have a thorough grasp of how each hyper-parameter affects the performance of the model. This iterative approach enables the development of an optimal architecture, maximizing translation accuracy through the application of the most effective hyper-parameter set.

III.3.2 Minimal Transformer Model Tuning

To identify the most effective optimizer for our hyper-parameter exploration, we compared four commonly used optimizers on an arbitrary minimal model. This model comprises a single layer, a feed-forward dimension of 256, an embedding dimension of 32, one attention head, a label smoothing of 0.6, and a dropout rate of 0.3. All optimizers were configured with the same learning rate of 1.

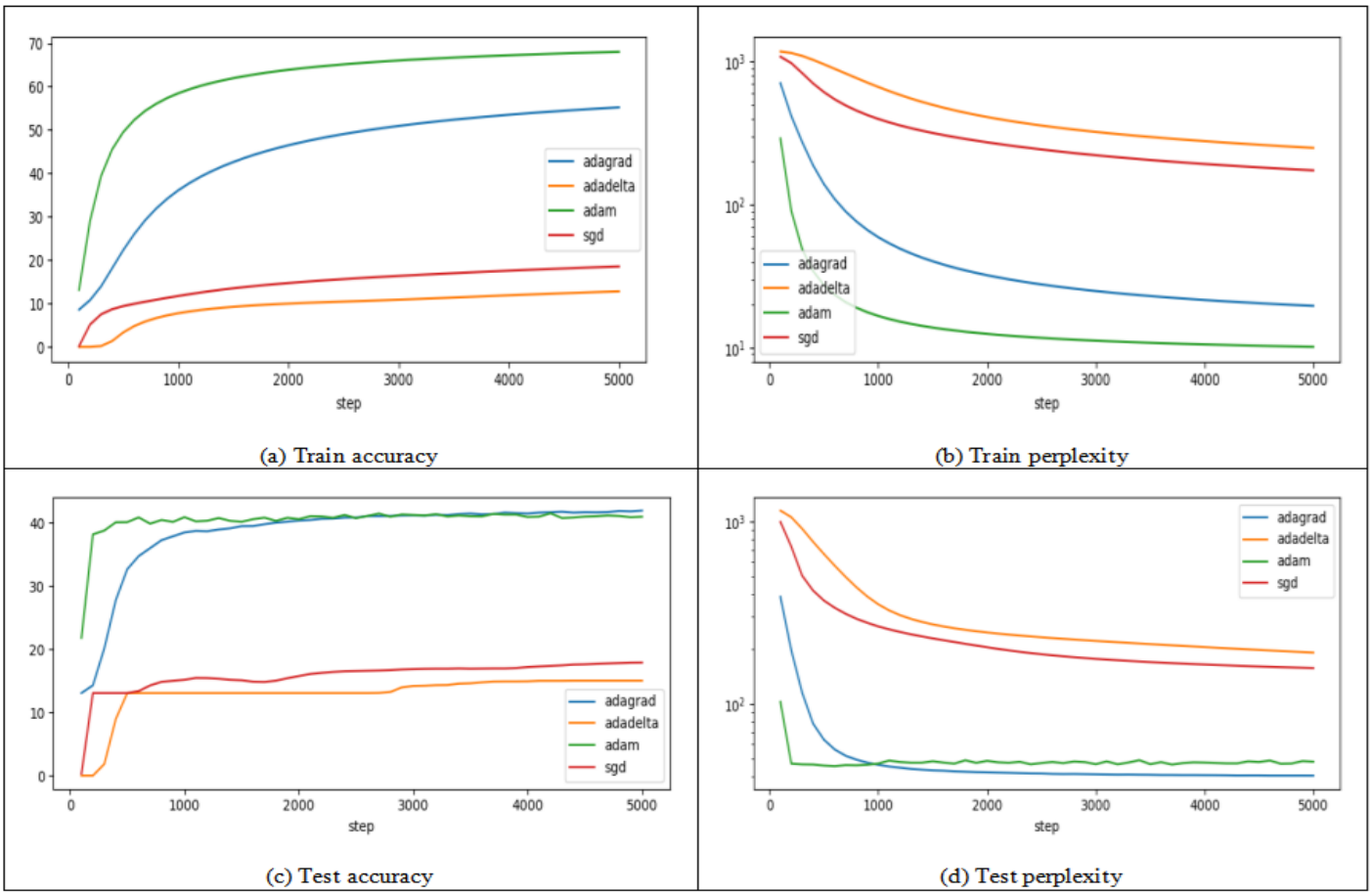

Figure 3: Comparative performance of stochastic optimizers on train and test sets for the minimal transformer model.
Source: Authors, (2025).

As can be observed in Figure 3, SGD and AdaDelta had the worst performance, as they had the lowest train and test accuracy, and the highest train and test perplexity. A logarithmic scale is used on the perplexity plots to facilitate the visualization of their progress, which was particularly necessary due to their significantly poorer performance. Adam and AdaGrad had the best results with Adam taking the lead with a higher train accuracy and a lower train perplexity. And AdaGrad had a marginally higher test accuracy and lower test perplexity.

Table 4: Comparison of ROUGE and BLEU scores for different optimizers on the minimal transformer model.

OPTIMIZER	ROUGE	BLEU			
		BLEU-1	BLEU-2	BLEU-3	BLEU-4
ADADELTA	1.75	16.5	0.0	0.0	0.0
SGD	12.08	22.7	4.8	3.3	0.0
ADAGRAD	48.14	**55.6**	**23.3**	**11.5**	**6.3**
ADAM	**48.78**	54.2	22.5	11.0	5.5

Source: Authors, (2025).

We report the best ROUGE and best BLEU-1 score accompanied by the corresponding BLEU-2, BLEU-3, and BLEU-4 scores of the resulting model's performances for each optimizer in Table 4. SGD and AdaDelta have the poorest performance with AdaDelta having a BLEU-2, BLEU-3, and BLEU-4 score of 0, and SGD having a BLEU-4 of 0.

This indicates that the model trained with the SGD optimizer struggles to generate 4-gram sequences that match a 4-gram sequence existing in the testing corpus, while the same can be said about the model trained using the AdaDelta optimizer, but in addition to 4-grams, the AdaDelta model fails to get matching 3-gram and 2-gram sequences too.

AdaGrad takes the lead when it comes to BLEU scores with BLEU-1, BLEU-2, BLEU-3, and BLEU-4 of 55.6, 23.3, 11.5, and 6.3 respectively, but Adam has an apparent advantage in terms of ROUGE scores with a 48.74 score. Stemming from these results, we decided to adopt the Adam optimizer for training our models and performing our hyper-parameters optimization.

Table 5: Dropout hyper-parameter exploration metrics.

METRICS	DROPOUT				
	0.1	0.2	0.3	0.4	0.5
BLEU-1	61.2	62.6	63.6	63.1	62.9
STEP	700	700	1000	7600	1000
ACCURACY	45.76	47.17	47.35	47.9	45.6
ROUGE	54.41	53.79	55.18	52.78	55.55

Source: Authors, (2025).

Table 6: Attention heads hyper-parameter exploration metrics.

METRICS	ATTENTION HEADS			
	1	2	4	8
BLEU-1	62.5	63.6	63.1	61.4
STEP	800	1000	1500	2900
ACCURACY	47.53	47.35	47.72	47.88
ROUGE	54.22	55.18	54.56	53.96

Source: Authors, (2025).

III.3.3 Fine-tuning Our Proposed Transformer Model

With the optimal optimizer identified in the first experiment, we shifted our focus to constructing our Transformer-based text-to-GLOSS model, in this subsection, we shed light on the consecutive hyper-parameter exploration to then unveil our final architecture. Our consecutive hyper-parameter exploration was performed manually and using the BLEU-1 metric for scoring the models and picking the best hyper-parameter in each run. Table 5 presents the dropout BLEU-1 score of our last hyper-parameter exploration run, in addition to the best BLEU-1 score of each dropout, the table also shows the step at which the score was obtained as well as the accuracy and the ROUGE score at that step. The table reveals that despite having a slightly lower accuracy on the test set, the dropout value of 0.3 yields the best BLEU-1 score of 63.6, and reaches its optimal performance in the 1000th step. The results for the attention tuning run are also provided in Table 6, a similar trend can be observed in the attention heads tuning run where despite not having the best accuracy, the model with 2 attention heads still yields a better BLEU-1 score of 63.6 taking the lead in the attention tuning run, it's best performance was achieved at the 1000th step, with an accuracy of 47.35.

The hyper-parameter optimization process took place until the model's parameters settled at the same value. The final hyper-parameter set is depicted in Table 7. The final model reached a training accuracy of 77.21, which translates to 47.35 test accuracy. It also reached BLEU scores of 63.6, 28.5, 15.2, and 9.0 in BLEU-1, BLEU-2, BLEU-3, and BLEU-4 respectively, and a ROUGE score of 55.18. The best BLEU-1 score was obtained at the 1000th training step, while the best ROUGE score was obtained at the 4900th step

Once our best-performing parameters were reached, the final parameter set was used to perform an optimizer comparison. Figure 4 illustrates the achieved performance metrics of the final model using the four optimizers: Adam, AdaGrad, AdaDelta, and SGD. The optimizers had the same parameters as the first optimizers trial in conjunction with the model's parameter set in Table 7. When it comes to the train set performance, Adam is clearly ahead of AdaGrad in both accuracy and perplexity. Furthermore, even with AdaGrad having a closer performance to Adam on the test set, Adam still takes the lead with a higher test accuracy, and a lower test perplexity. Both SGD and AdaDelta have significantly worse performance compared to Adam and AdaGrad over both the train and the test set.

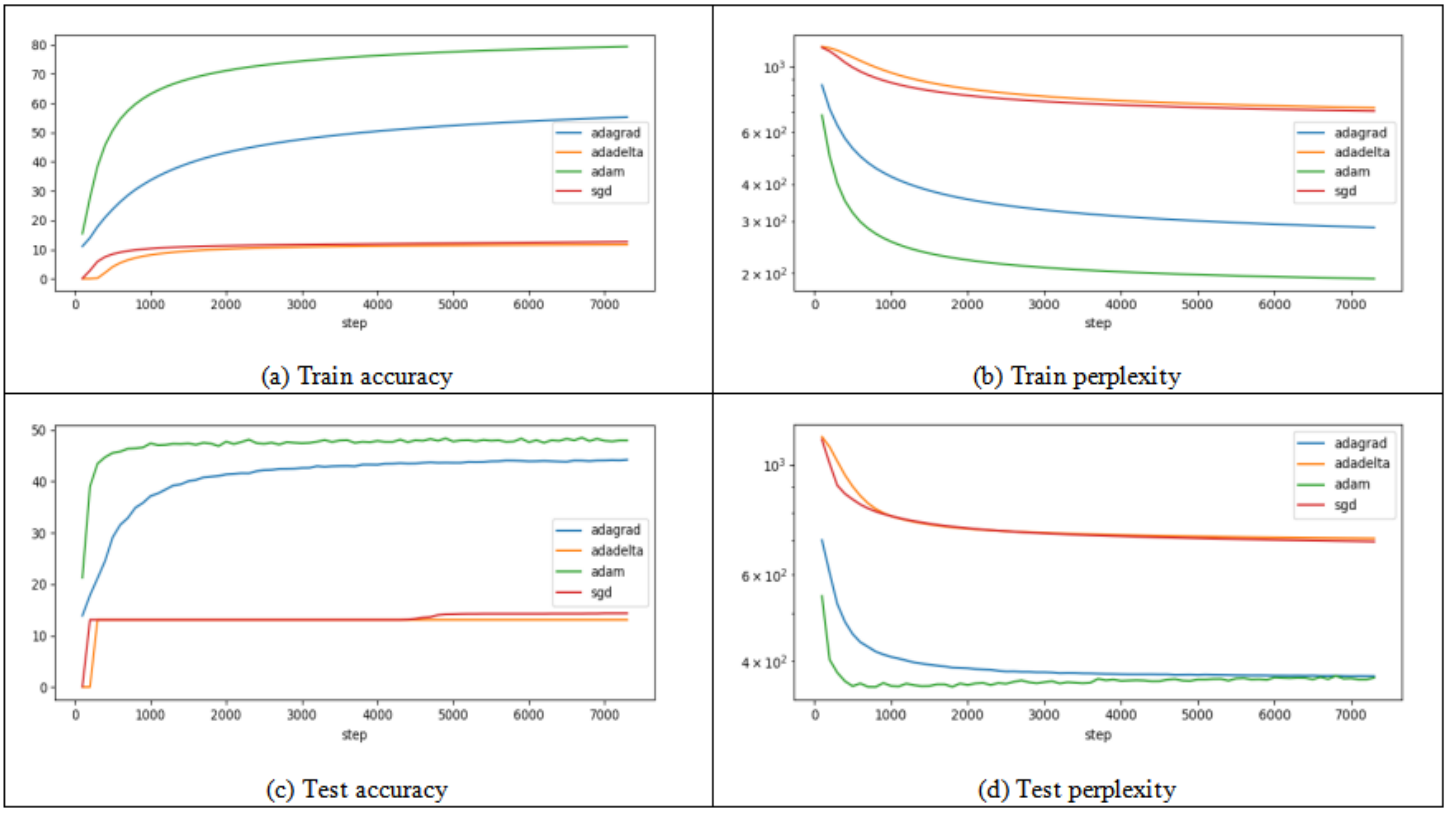

Figure 4: Comparative performance of stochastic optimizers on train and test sets for our proposed transformer model.
Source: Authors, (2025).

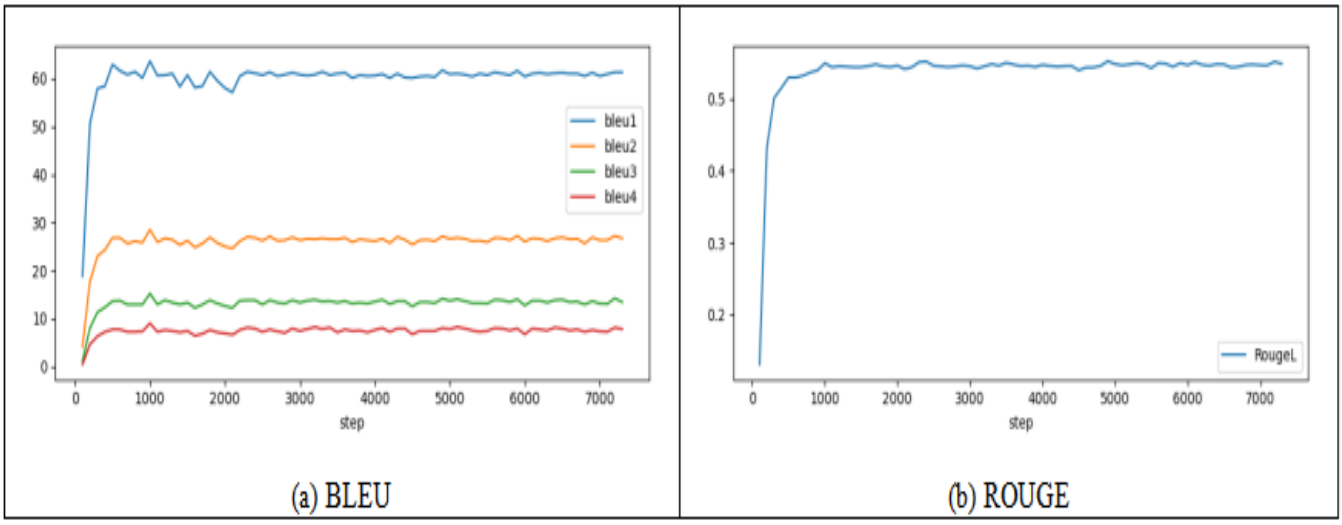

Figure 5: Performance metrics evolution of our proposed transformer-based model for text-to-GLOSS translation during training.
Source: Authors, (2025).

Table 7: Optimal hyper-parameters for our proposed transformer model.

Hyper-parameter	Value
Warmup steps	300
Batch-size	4096
Attention heads	2
Number of layers	5
Embedding dimension	64
Feed-forward dimension	256
Dropout	0.3
Label smoothing	0.6

Source: Authors, (2025).

Table 8 presents the best ROUGE scores obtained from the experiment and the best BLEU-1 score with the corresponding BLEU-2 to 4 scores. From the results presented in the table, we can observe that the AdaDelta optimizer has the worst scores, with 0.09 ROUGE and 0.0 in all BLEU scores. The SGD optimizer has slightly better scores with 4.10 ROUGE, 13.5 BLEU-1, and 0.0 BLEU-2, BLEU-3, and BLEU-4. AdaGrad takes the second position with 50.72 ROUGE, 57.8 BLEU-1, 23.3 BLEU-2, 11.6 BLEU-3, and 6.4 BLEU-4. Finally, the best-performing optimizer is Adam with a significantly better score than AdaGrad. It has a ROUGE score of 55.18, and its BLEU scores were 63.6, 28.5, 15.2, and 9.0 for the BLEU-1, BLEU-2, BLEU-3, and BLEU-4 respectively. The results of the experiment clearly show that the Adam optimizer is significantly better for our specific use-case of text-to-GLOSS neural machine translation using a Transformer architecture.

Table 8: Comparison of ROUGE and BLEU scores for different optimizers in our proposed transformer model.

OPTIMIZER	ROUGE	BLEU			
		BLEU-1	BLEU-2	BLEU-3	BLEU-4
ADADELTA	0.09	0.0	0.0	0.0	0.0
SGD	4.10	13.5	0.0	0.0	0.0
ADAGRAD	50.72	57.8	23.3	11.6	6.4
ADAM	**55.18**	**63.6**	**28.5**	**15.2**	**9.0**

Source: Authors, (2025).

III.3.4 Performance Evaluation and Comparative Analysis

Figure 5 shows the evolution of the ROUGE, BLEU, and BLEU-1 to 4 performance metrics for our proposed model with Adam optimizer on the test set during training. we can notice that the models improve the most in the first thousand or so training steps, then the performance only varies slightly in each evaluation, these variations are more pronounced in the BLEU scores than the ROUGE score.

Table 9: Comparison of ROUGE and BLEU scores for different model architectures.

ARCHITECTURE	ROUGE	BLEU			
		BLEU-1	BLEU-2	BLEU-3	BLEU-4
CNN	49.91	59.8	25.4	13.4	8.4
LSTM	46.27	51.1	17.8	7.3	3.3
GRU	25.90	31.0	5.7	0.9	0.2
OUR PROPOSED MODEL	**55.18**	**63.6**	**28.5**	**15.2**	**9.0**

Source: Authors, (2025).

To compare our system's performance with other architectures, we built several models with different architectures. All the architectures were built using the default configuration of the OpenNMT-py for translation. Three architectures were constructed for this comparison: a CNN [29] model, an LSTM [30] model, and a GRU [31] model. All the models have two encoder layers, and two decoder layers, a hidden size of 500, and optimized using the SGD optimizer. The CNN has a kernel width of 3. Table 9 presents the BLEU and ROUGE scores of all the evaluated methods, out of the three tested architectures, the CNN takes the first position with a ROUGE score of 49.91, and a BLEU-1, BLEU-2, BLEU-3, and BLEU-4 of 59.8, 25.4, 13.4, and 8.4 respectively. However, the table clearly demonstrates the transformer architecture taking a lead in all scores with a 55.18 ROUGE score, and 63.6, 28.5, 15.2, and 9.0 for the BLEU-1, BLEU-2, BLEU-3, and BLEU-4 respectively.

Table 10: Comparison of ROUGE and BLEU scores for our proposed approach with state-of-the-art methods on PHOENIX14T corpus test set.

METHODS	ROUGE	BLEU			
		BLEU -1	BLEU -2	BLEU -3	BLEU -4
RNN WITH LUONG ATTENTION [7]	48.10	50.67	32.25	21.54	15.25
GRU WITH BAHDANAU ATTENTION [8]	42.96	43.90	26.33	16.16	10.42
TRANSFORMER [9]	54.55	55.18	**37.10**	**26.24**	**19.10**
OUR PROPOSED APPROACH	**55.18**	**63.6**	28.5	15.2	9.0

Source: Authors, (2025).

In Table 10 our model's performance is compared to previous studies. The PHOENIX-14T text-to-GLOSS dataset is used to obtain the results. To provide a comprehensive overview of the translation performance, both the best ROUGE score, and the BLEU-1 to BLEU-4 scores are provided. Additionally, the architecture of each system is provided. The highest BLEU score achieved by our model is 19.04. Given that the BLEU-1 score is employed in our hyper-parameter exploration, it exhibited the most significant performance improvement when compared to alternative systems. In regards to the BLEU-1 score, our model outperforms all other approaches with a significant increase of 19.7 compared to [8], our model also outperforms the models suggested in [7] and [9] by 12.93 and 8.42, respectively. For BLEU-2, BLEU-3, and BLEU-4 scores, our model's performance is on par with the model in [8], with scores of 28.5, 15.2, and 9.0, respectively. Our system outperforms both the GRU with attention proposed in [8] and the Recurrent Neural Network (RNN) based architecture with attention proposed in [7], with ROUGE score increases of 12.22 and 7.08, respectively. Our model also achieves a 0.63 improvement in ROUGE score, marginally outperforming the Symbolic Transformer suggested in [9].

III.3.5 Performance Evaluation and Comparative Analysis.

In this subsection we present the benchmarking results for our model's inference using CTranslate2 on both CPU and GPU. We evaluate the model's inference performance based on three metrics: time per token, sentence latency, and memory usage. Figure 6 illustrates the comparison of these metrics across bot GPU and CPU implementations.

The results reveal that the average time per token on the CPU is 0.28 milliseconds, which is approximately three times lower than the 0.86 millisecond observed on the GPU. Similarly sentence latency on the CPU averages 1.69 milliseconds, whereas on the GPU it is approximately three times higher with 5.09 milliseconds. Additionally, the CPU's model memory usage is around 8.37 MB, which is slightly lower than the 10MB recorded on the GPU. These figures provide a clear comparison of the performance between CPU and GPU implementations of the model during inference.

IV. DISCUSSION

Our study aims to investigate the performance of optimizers in the context of finding the most optimal transformer architecture for the real-time text-to-GLOSS translation task. We hypothesize that by initially exploring optimizers using a minimal model and subsequently applying the insights gained to optimize a more complex transformer architecture, we can obtain more insights into optimizer performance in the text-to-GLOSS translation task across different scenarios.

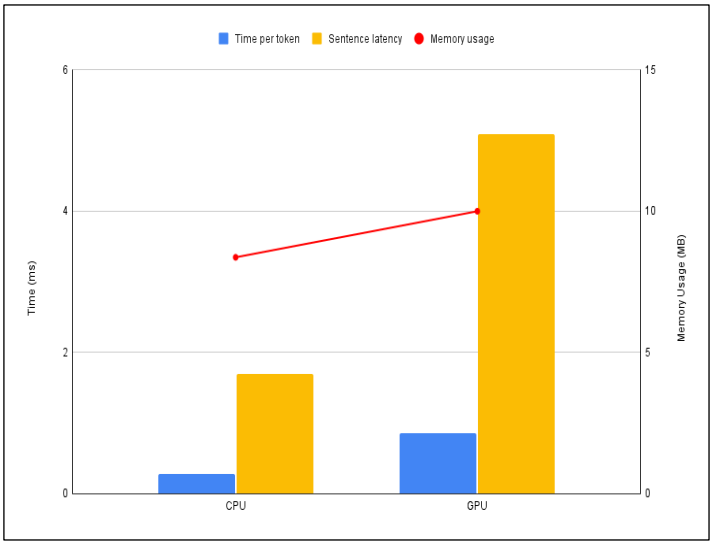

Figure 6: CPU and GPU performance metrics during model inference.
Source: Authors, (2025).

Our study employs a two-phase experimental approach. In the first phase, we conduct a comprehensive exploration of optimizers using a minimal transformer model. The goal is to identify the most effective optimizer through a comparative evaluation of AdaDelta, SGD, AdaGrad, and Adam. Accordingly, our model of choice featured a single feed-forward layer with a dimension of 256, an embedding dimension of 32, one attention head, a dropout rate of 0.3, and a label smoothing of 0.6. All optimizers were configured with a learning rate of 1. This first phase yielded significant insights into the performance and adaptability of the optimizers on a minimal model in this specific use case. Both AdaDelta and SGD resulted in a model with subpar performance, with a ROUGE score of 1.75 and 12.08 respectively. However, the AdaGrad and Adam optimizers achieved superior results with 48.14, and 48.78 ROUGE respectively. Despite AdaDelta being an extension of AdaGrad, it yields significantly inferior performance in our specific testing conditions. The observed performance discrepancy may stem from AdaDelta's limited adaptability to the intricacies of our text-to-GLOSS translation task, suggesting potential challenges in generalizing its optimization capabilities for this specific context. Subsequently, the findings obtained from the optimizer screening phase guide the subsequent hyper-parameter exploration, aiming to identify the optimal transformer architecture in the second phase.

Our investigative approach emphasizes a sequential optimization process. Initially, we analyze the optimizers in a simpler context and subsequently apply the insights to guide the hyper-parameter exploration for identifying the optimal architecture. The hypothesis underscores the critical role of a well-suited optimizer in attaining optimal convergence and ensuring

high translation quality, particularly in the complex task of text-to-GLOSS translation using a transformer model.

To enhance the search for the optimal architecture, we leverage the insights from the optimizer screening to guide the exploration of hyper-parameters and model configurations. Employing a consecutive hyper-parameter exploration technique for Transformers. This process is used to address the excessively intensive resource demand of an exhaustive parameter exploration and the suboptimal performance obtained from a random parameter search. Our optimization process starts with the selection of a primary hyper-parameter set, with subsequent sequential optimization of each parameter. During this process, one hyper-parameter is altered at a time while maintaining the others fixed, evaluating the model's translation quality on the selected corpus. Following that, hyper-parameter values are modified in accordance with the outcomes of the first optimization run. This continual process of fine-tuning proceeds for each hyper-parameter individually until the model's output no longer shows any signs of progress.

In addition to comparing optimizers within the context of the minimal model, our analysis extends to a larger and more intricate transformer architecture. This broader comparison serves the purpose of affirming the robustness and consistency of the selected optimizer, ensuring that the insights obtained from the initial experiment are applicable to a practical, real-world text-to-GLOSS translation scenario. The increased complexity of the larger model accentuates the performance distinctions observed with each optimizer. AdaDelta and SGD yield ROUGE scores of 0.09 and 4.10, respectively, significantly lower than the scores achieved in the smaller model. This outcome further reinforces our hypothesis regarding the critical role of a well-suited optimizer tailored to specific conditions. In contrast, AdaGrad demonstrates a ROUGE score of 50.72, while Adam excels with a ROUGE score of 55.18. This comparison within the larger model context reinforces the reliability of the identified optimizer in delivering high-quality results across diverse scenarios.

In the study by Choi et al. [32], the authors observed significant variability in optimizer performance depending on the workload. While some workloads exhibited comparable performance across all tested optimizers, in other scenarios, there were substantial differences leading to clear distinctions in both predictive performance and training speed. Notably, the efficiency of Adam was particularly evident, requiring significantly fewer training steps than SGD to achieve the same target error on a transformer architecture. Our findings resonate with this observed performance difference, as SGD demonstrated notably reduced effectiveness for our specific text-to-GLOSS translation task compared to Adam. These results underscore the critical importance of carefully selecting the appropriate optimizer tailored to the unique characteristics of each workload or task. Several studies have suggested that the suboptimal performance of SGD on attention models can be attributed to heavy-tailed noise, as noted in Zhang et al.'s work [33]. They propose that Adam's success in optimizing these models is linked to its resilience against outliers. However, Chen et al. [34] challenge this notion. Backed by controlled stochasticity experiments through varied batch sizes, the study proposes that stochasticity and heavy-tailed noise might not be significant contributors to the observed performance discrepancy. Instead, it suggests that Adam-like methods utilize a descent direction that is superior to the gradient, providing an alternative explanation for their effectiveness. Using a consecutive hyper-parameter exploration, we managed to find an optimal Transformer architecture and significantly increase the translation performance over the PHOENIX-14T dataset highlighting the importance of finding a task-specific parameter set for achieving a significant performance. This aligns with Araabi et al.'s study [10] that shows through experimental evidence the performance increase of a properly configured Transformer for low-resource language conditions.

Following the optimizer comparison, we performed real-time benchmarking to evaluate the model's performance in practical scenarios. The benchmarking results revealed that the CPU outperformed the GPU in terms of time per token and sentence latency for small models and short sentences. Specifically, the CPU's time per token was approximately three times lower than that of the GPU, and sentence latency on the CPU was about three times faster. Additionally, the CPU's memory usage was slightly lower than that of the GPU. These findings suggest that for small models, CPUs offer a more efficient solution compared to GPUs.

In the context of real-time inference, our results are consistent with [35], who also observed better performance with inference on CPU compared to GPU despite having a CPU with less peak FLOP performance. Wu et al. reported that decoding their model on CPU was 2.3 times faster than on GPU. They attributed this discrepancy to the significant overhead caused by non-trivial amount of data transfer between the host and the GPU at every decoding step.

Overall, our findings provide a comprehensive understanding of the optimization strategies and hardware considerations crucial for maximizing the performance of neural machine translation models. The evaluation of different optimizers and hyper-parameter settings has revealed significant performance gains, while the real-time benchmarking highlights the importance of hardware choice, particularly the efficiency of CPUs for specific tasks. These insights align with and extend existing research, and contribute valuable knowledge to the field, guiding future research and practical implementations.

V. CONCLUSIONS

This paper presents a novel transformer-based Neural Machine Translation model specifically tailored for real-time text-to-GLOSS translation. First, we provided a comprehensive exploration of optimizers to identify the most optimal transformer architecture for the text-to-GLOSS translation task. The initial phase involved a comprehensive examination of optimizers using a minimal transformer model. This phase revealed significant variations in performance, with Adam emerging as a robust choice for our specific use case reaching 48.78 ROUGE. Building upon the optimizer screening phase, our consecutive hyper-parameter exploration is used to fine-tune the search for the optimal transformer architecture. The iterative process, sequentially refining each hyper-parameter, supported our exploration of the complex landscape of model configurations. This methodological refinement proved essential in identifying an architecture that significantly enhanced text-to-GLOSS translation performance over the PHOENIX-14T dataset. The comparison of optimizers extended to a larger and more intricate transformer architecture, affirming the robustness of our selected optimizer, Adam, across diverse scenarios. The performance distinctions observed in the larger model context reinforced the hypothesis that the choice of the optimizer, coupled with the right hyper-parameter set, plays a pivotal role in achieving optimal convergence and translation quality, particularly in complex tasks.

Furthermore, we show that our obtained model using these techniques not only outperforms alternative architectures such as CNN, LSTM, and GRU in both ROUGE and BLEU 1 to 4 scores, but also outperforms state-of-the-art models not only on the optimization target metric (BLEU1), but also on the ROUGE metric, setting a new benchmark for text-to-GLOSS translation on the PHOENIX-14T dataset with 63.6 BLEU1 and 55.18 ROUGE scores further establishing the significance of our findings in the field. In terms of real-time inference, our benchmarking results indicate that for small models, the CPU significantly outperforms the GPU, with the CPU achieving approximately three times lower time per token and three times faster sentence latency. These insights emphasize the importance of hardware considerations in deployment, as optimizing for real-time performance can greatly enhance the practical applicability of NMT systems. Our findings hold great promise for diverse applications, ranging from education to healthcare, offering enhanced accessibility through real-time sign language translation for the Deaf and hard-of-hearing community. By addressing specific challenges in sign language translation, particularly the need for real-time processing, our research paves the way for seamless and uninterrupted communication, significantly improving inclusivity for the DHH community.

VI. ACKNOWLEDGMENTS

We extend our sincere gratitude to Imane Lasri for their invaluable comments and insightful suggestions that greatly enhanced the quality and rigor of this article.

VII. AUTHOR CONTRIBUTIONS

Conceptualization: Younes Ouargani, Noussaima El Khattabi.
Methodology: Younes Ouargani.
Investigation: Younes Ouargani.
Discussion of results: Younes Ouargani, Noussaima El Khattabi.
Writing – Original Draft: Younes Ouargani.
Writing – Review and Editing: Younes Ouargani.
Resources: Younes Ouargani.
Supervision: Noussaima El Khattabi.
Approval of the final text: Younes Ouargani, Noussaima El Khattabi.

VIII. REFERENCES

[1] M. Kay, “The proper place of men and machines in language translation,” *machine translation*, vol. 12, pp. 3–23, 1997.

[2] P. F. Brown *et al.*, “A statistical approach to machine translation,” *Computational linguistics*, vol. 16, no. 2, pp. 79–85, 1990.

[3] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014.

[4] A. Vaswani *et al.*, “Attention Is All You Need,” *arXiv:1706.03762 [cs]*, Dec. 2017.

[5] R. San-Segundo *et al.*, “Speech to sign language translation system for Spanish,” *Speech Communication*, vol. 50, no. 11, pp. 1009–1020, Nov. 2008, doi: 10.1016/j.specom.2008.02.001.

[6] A. Almohimeed, M. Wald, and R. I. Damper, “Arabic Text to Arabic Sign Language Translation System for the Deaf and Hearing-Impaired Community,” in *Proceedings of the Second Workshop on Speech and Language Processing for Assistive Technologies*, N. Alm, Ed., Edinburgh, Scotland, UK: Association for Computational Linguistics, Jul. 2011, pp. 101–109. Accessed: Jan. 25, 2024. [Online]. Available: https://aclanthology.org/W11-2311

[7] S. Stoll, N. C. Camgoz, S. Hadfield, and R. Bowden, “Text2Sign: Towards Sign Language Production Using Neural Machine Translation and Generative Adversarial Networks,” *International Journal of Computer Vision*, vol. 128, no. 4, pp. 891–908, Apr. 2020, doi: 10.1007/s11263-019-01281-2.

[8] M. Amin, H. Hefny, and A. Mohammed, “Sign Language Gloss Translation using Deep Learning Models,” *International Journal of Advanced Computer Science and Applications*, vol. 12, Jan. 2021, doi: 10.14569/IJACSA.2021.0121178.

[9] B. Saunders, N. C. Camgoz, and R. Bowden, “Progressive Transformers for End-to-End Sign Language Production,” in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 687–705. doi: 10.1007/978-3-030-58621-8_40.

[10] A. Araabi and C. Monz, “Optimizing Transformer for Low-Resource Neural Machine Translation,” in *Proceedings of the 28th International Conference on Computational Linguistics*, Barcelona, Spain (Online): International Committee on Computational Linguistics, Dec. 2020, pp. 3429–3435. doi: 10.18653/v1/2020.coling-main.304.

[11] W. Hao, H. Xu, L. Mu, and H. Zan, “Optimizing Deep Transformers for Chinese-Thai Low-Resource Translation,” in *Machine translation*, 2022, pp. 117–126. doi: 10.1007/978-981-19-7960-6_12.

[12] Y. Ouargani and N. E. Khattabi, “Advancing text-to-GLOSS neural translation using a novel hyper-parameter optimization technique.” 2023. Available: https://arxiv.org/abs/2309.02162

[13] M. D. Zeiler, “ADADELTA: An Adaptive Learning Rate Method,” *arXiv.org*. Dec. 2012.

[14] J. Kiefer and J. Wolfowitz, “Stochastic Estimation of the Maximum of a Regression Function,” *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952.

[15] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization.” arXiv, Jan. 2017. doi: 10.48550/arXiv.1412.6980.

[16] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization.” *Journal of machine learning research*, vol. 12, no. 7, 2011.

[17] K. Cho *et al.*, “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.” arXiv, Sep. 2014.

[18] D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate.” arXiv, May 2016.

[19] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[20] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer Normalization.” arXiv, Jul. 2016.

[21] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: A Method for Automatic Evaluation of Machine Translation,” in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, P. Isabelle, E. Charniak, and D. Lin, Eds., Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Jul. 2002, pp. 311–318. doi: 10.3115/1073083.1073135.

[22] C.-Y. Lin, “ROUGE: A Package for Automatic Evaluation of Summaries,” in *Text Summarization Branches Out*, Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81.

[23] G. Klein, F. Hernandez, V. Nguyen, and J. Senellart, “The OpenNMT Neural Machine Translation Toolkit: 2020 Edition,” in *Proceedings of the 14th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*, M. Denkowski and C. Federmann, Eds., Virtual: Association for Machine Translation in the Americas, Oct. 2020, pp. 102–109.

[24] G. Klein, Y. Kim, Y. Deng, V. Nguyen, J. Senellart, and A. M. Rush, “OpenNMT: Neural Machine Translation Toolkit.” arXiv, May 2018. doi: 10.48550/arXiv.1805.11462.

[25] A. Paszke *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Advances in Neural Information Processing Systems*, 2019.

[26] N. C. Camgoz, S. Hadfield, O. Koller, H. Ney, and R. Bowden, "Neural sign language translation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7784–7793.

[27] O. Koller, J. Forster, and H. Ney, "Continuous sign language recognition: Towards large vocabulary statistical recognition systems handling multiple signers," *Computer Vision and Image Understanding*, vol. 141, pp. 108–125, Dec. 2015, doi: 10.1016/j.cviu.2015.09.013.

[28] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization." *Journal of machine learning research*, vol. 13, no. 2, 2012.

[29] A. Waibel, "Phoneme recognition using time-delay neural network." 1989.
[30] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[31] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches." arXiv, Oct. 2014.

[32] D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, and G. E. Dahl, "On empirical comparisons of optimizers for deep learning." 2020. Available: https://arxiv.org/abs/1910.05446

[33] J. Zhang *et al*., "Why are adaptive methods good for attention models?" *Advances in Neural Information Processing Systems*, vol. 33, pp. 15383–15393, 2020.

[34] J. Chen, F. Kunstner, and M. Schmidt, "Heavy-tailed noise does not explain the gap between SGD and adam on transformers," in *13th annual workshop on optimization for machine learning*, 2021.

[35] Y. Wu *et al.*, "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation." arXiv, Oct. 2016. doi: 10.48550/arXiv.1609.08144.