university of groningen

faculty of science and engineering

Next-Word Prediction in Low-Resource Languages: A Study on Hrusso Aka

Master's Research Project

By:
Alina Dima

October 31, 2024

Supervised by:
Dr. Stephen Jones
Dr. Tsegaye Tashu
University of Groningen

Abstract

With most of the languages worldwide being endangered or potentially vulnerable, it is essential to develop technological tools that facilitate the preservation of these languages. This project focuses on Hrusso Aka, an endangered isolate language spoken in Northeast India. The aim is to contribute to the revitalisation of this language by developing a next-word prediction model for mobile device keyboards. Due to the limited written resources available in Hrusso Aka, the major challenge of this study is data scarcity. To develop a model that can learn effectively with such limited data, we experimented with models of varying complexity. This involved implementing and comparing an *n*-gram model, a recurrent neural network (RNN) model, and a compact transformer model. The models were evaluated on two criteria, namely, the accuracy of predicting the next word and the inference time, as the model should function in real time for mobile keyboard use. Based on our assessment, the GRU-based RNN model significantly outperformed the *n*-gram and transformer models on both evaluation criteria. The GRU-based RNN achieved a top-3 accuracy of 19.94%, and an inference time of 0.0006s, compared to the *n*-gram and transformer models which were slower and achieved a top-3 accuracy of less than 10%. These findings suggest that, although transformer models are state-of-the-art solutions for many high-resource tasks, simpler models appear to be better suited for low-resource languages.

Contents

1 Introduction

“The diversity of languages is not a diversity of signs and sounds but a diversity of views of the world.”

—*Wilhelm von Humboldt*

Language is more than a means of communication; it is a gateway into a culture’s environment and history. As Humboldt suggested in his first academic discourse from 1820 (Trabant, 2000), linguistic diversity is a reflection of cultural diversity itself. Thus, each language signifies a culture’s unique viewpoint that has evolved through time alongside its people. For many cultures that rely on oral tradition, the language and the stories that stood the test of time might be the only testaments of the generations that passed. In the context of language endangerment, especially in cultures with limited documentation of their history and traditions, linguistic extinction would not only result in the loss of unique words and sounds but also in the potentially irreversible deterioration of a substantial part of cultural identity.

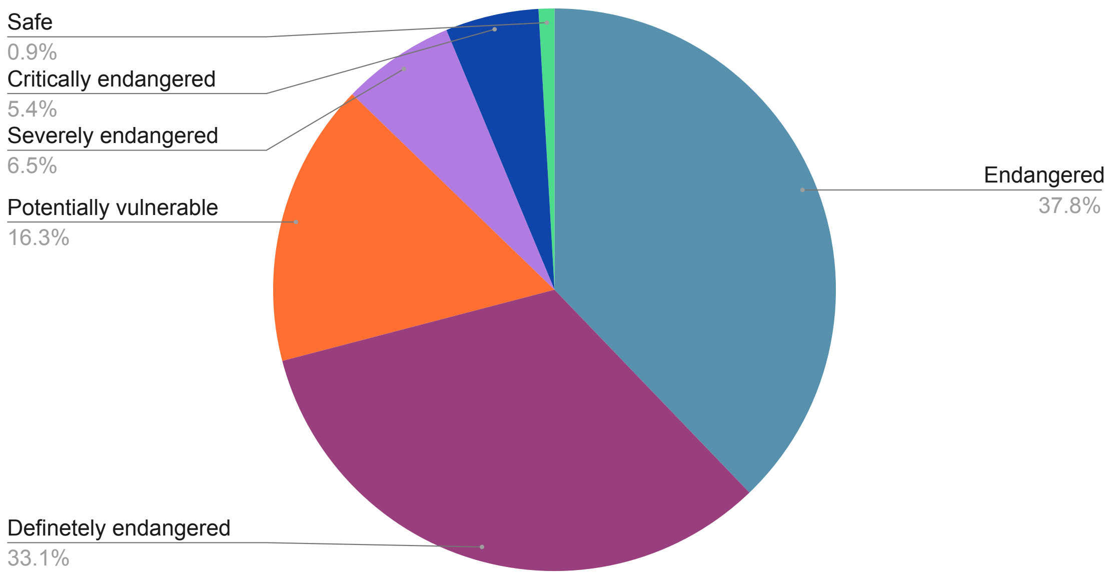


Figure 1: Language endangerment chart based on 8324 languages, out of which 7000 are still in use. The data was retrieved from (UNESCO, 2024).

Over the past decades, the urgency of addressing language endangerment has become clearer. Although the exact numbers differ based on source, there are approximately 7000 languages still in use, out of which at least half are considered endangered by multiple sources (Sutherland, 2003; Eberhard, Simons, & Fennig, 2019; UNESCO, 2024). The United Nations Educational, Scientific and Cultural Organization (UNESCO) paints one of the grimmest pictures, considering only around 1% of the languages safe, the others being either endangered or potentially vulnerable, which is to be observed in more detail in Figure 1. Other studies suggested that one language dies every one to three months (Crystal, 2000; Simons, 2019) and that language loss could triple in the next 40 years (Bromham et al., 2022), leading to the death of over 1500 languages by 2100.

Around the world, the number of language preservation efforts is increasing, ranging from language documentation initiatives (Endangered Languages Documentation Programme, 2024) to language learning programmes (7000 Languages, 2024). These initiatives are further supported by the widespread availability of technology and the advancements in natural language processing (NLP), facilitating direct engagement with e-learning platforms that leverage automated exercise generation and assessment (Zhang, Frey, & Bansal, 2022; Katinskaia, Nouri, & Yangarber, 2017).

The field of NLP has surged in popularity in recent years both in academia and in industry. The field began with statistical approaches such as *n*-grams proposed in the 1980s (Jelinek & Mercer, 1980) and popularised in the 1990s on speech recognition tasks (Jelinek, 1998). Since then NLP has evolved into Recurrent Neural Network (RNN) models able to capture long-term dependencies (Hochreiter & Schmidhuber, 1997; Cho, Van Merriënboer, Gulcehre, et al., 2014), and then into complex transformer architectures able to tackle challenging tasks such as language generation, question-answering or summarisation (Devlin, Chang, Lee, & Toutanova, 2018; Radford, Narasimhan, Salimans, Sutskever, et al., 2018). The computational advancements and the continuous increase in the amount of data have led to efficient models that made their way into the lives of many of us through interactions with services such as ChatGPT (OpenAI, 2022). The field of NLP, as well as the implementation of the approaches themselves, are discussed in more detail in Section 2.2.

The number of NLP papers has increased almost exponentially since 2017 (Chen, Xie, & Tao, 2022). However, most of these advancements are concentrated on data coming from only 20 languages (Magueresse, Carles, & Heetderks, 2020) for which large volumes of data are easily available. Consequently, the majority of languages are underrepresented. The disparity is even more pronounced when it comes to endangered languages, which are often classified as very low-resource languages. Such languages are seldom studied which exacerbates the knowledge gap and further alienates people from regular engagement with their mother tongue in today's highly digital world. An overview of the current research landscape of NLP in very low-resource languages is presented in Section 2.3.

1.1 About This Project

This project focuses on the Hrusso Aka language, one of the endangered languages in Northeast India (Moseley, 2010). Currently, the language is spoken by approximately 3000 people (van Driem, 2007) and faces the threat of extinction without active preservation efforts. Multiple socioeconomic factors, further discussed in Section 2.1, have contributed to the decline in Hrusso Aka usage, especially among the younger generations.

To contribute towards the preservation and revitalisation of Hrusso Aka, we propose a next-word prediction model, integrated into a mobile keyboard. This approach can blend into the lives of the youth, who are already highly connected to the digital world. By becoming part of people's daily lives, the keyboard would encourage active and regular engagement with the language.

NLP models evolved during the time of technological advancements in computer hardware alongside an increasing volume of data. As a result, many architectures were not designed with low-resource contexts in mind. One goal of this thesis is to compare models ranging in complexity from *n*-grams to transformers in a next-word prediction task for the very-low resource language Hrusso Aka. Since the best model will be integrated into a mobile keyboard, it is essential that during evaluation we balance the high accuracy usually offered by more complex models with real-world constraints such as processing time or memory, where simpler models may be preferred.

Focusing on an endangered language not only shifts the conversation towards the pressing matter of language preservation but also highlights the unique challenges posed by data scarcity. Thus, this project contributes to the field of NLP for very low-resource languages, which is currently such an underresearched area of artificial intelligence.

In a broader sense, the unique and limited environment that low-resource languages provide might inspire breakthroughs, leading to approaches that could potentially achieve state-of-the-art performance while being more efficient and sustainable in terms of computational resources and infrastructure.

1.1.1 Research Questions

To summarise, this thesis aims to develop a next-word prediction model for mobile devices in the language of Hrusso Aka. By integrating the model into a mobile keyboard, we aim to promote language engagement, especially among younger speakers of Hrusso Aka who are less prone to speak their native tongue. Moreover, by comparing different NLP approaches, our objective is to address the challenges caused by data scarcity and to highlight any potential differences in performance or approach compared to high-resource languages. We also aim to maximise model performance while prioritising the efficiency and user-friendliness of the mobile application. Consequently, this project has two main goals:

1. Answering the primary research question:

   What are the comparative performances of statistical models, RNN models and transformers in Hrusso Aka next-word prediction?

   and the secondary research question:

   What are the scalability differences between statistical models, RNNs, and transformer models in real-world use?

   To address the primary research question, we will implement three types of NLP models: n-gram, recurrent neural network (RNN), and transformer models. The models will be trained, fine-tuned, and, finally, evaluated based on their next-word prediction accuracy.

   To address the secondary research question, we should consider the user experience and real-world application. Because in practical scenarios memory is often constrained and users expect prompt responses, the trade-off between accuracy and computational efficiency constitutes a powerful metric for determining the feasibility of deploying a particular model.

2. Deploying the best-performing next-word prediction model into a mobile keyboard application.

1.1.2 Thesis Outline

The subsequent sections are organised as follows. Section 2 presents a comprehensive literature review of the Hrusso Aka language, NLP models, and application design principles. Section 3 details the Hrusso Aka dataset used for training and the methodology used to develop the next-word prediction model. Section 4 presents the results which are further discussed in Section 5, where current project limitations and future research directions are also proposed.

2 Theoretical Background

This section covers the key areas for developing and deploying next-word prediction models, particularly in the very low-resource language Hrusso Aka.

The section begins with an overview of Hrusso Aka, followed by an overview of relevant Natural Language Processing (NLP) methods. Next, an overview of NLP research in (very) low-resource languages is presented, followed by mobile application design principles and a description of federated learning.

2.1 Hrusso Aka

Hrusso Aka (also known as Ĝusso, Hrus(s)o, Aka-Hruso, Aka, Angka(e), or Tenae) is a language spoken by about 3000-4000 people in about 30 villages (van Driem, 2007; D'Souza, 2021a) in the West Kameng District of Arunachal Pradesh, India.

The rest of this subsection is dedicated to providing an overview of the linguistic affiliation and possible dialects of this language, its linguistic features, and the sociolinguistic factors that have led to its endangerment.

2.1.1 Linguistic Affiliation

The linguistic affiliation of Hrusso Aka has been debated since the early 1900s. The language was first classified into a linguistic family by Konow (1902), who considered it a non-Sinitic Sino-Tibetan language while acknowledging its divergence from other such languages regarding its phonetics. This view was later shared by Shafer (1947), who acknowledged possible phonetic similarities while arguing against vocabulary similarities. However, Shafer's publication has since been labelled problematic due to the misidentification of two dialects of Hrusso Aka, referred to as dialect A and dialect B. The possibility of two dialects has since been refuted by Blench and Post (2011, 2014) and D'Souza (2021a), who stated that what Shafer (1947) identified as dialect A is a different language called Sajolang (Miji).

In recent years, Blench and Post (2011, 2014) considered Hrusso Aka either a language isolate or a direct descendent of Proto-Sino-Tibetan without other links to other non-Sinitic Sino-Tibetan languages. Blench and Post (2011, 2014) consider the similarities to Sino-Tibetan languages weak and an effect of regional borrowing. Instead, Bodt and Lieberherr (2015) disagree and classify the language as a Sino-Tibetan language distantly related to Sajolang (Miji) and Levai (Bangru).

Overall, no consensus has been reached on the linguistic affiliation of Hrusso Aka. Even a possible affiliation would not be strongly supported and suggests weak similarities to other languages. Thus, for the rest of this work, we will treat Hrusso Aka as a language isolate.

2.1.2 Sociolinguistic Context of Endangerment

In their study of Hrusso Aka endangerment, Sinha and Barbora (2021) identified several factors that have led to the endangerment of this language. Firstly, the population increasingly encounters Hindi and English in church or when interacting with teachers, government officials, or workers from other parts of India. Additionally, Sinha and Barbora (2021) argues that intermarriages between different linguistic communities have become common, and since the indigenous languages do not share much similarity, the primary language used in these households often becomes Hindi. Lastly, as an effect of globalisation, many people, especially the youth, are immersed in Hindi or Western media.

As a result of the aforementioned factors, Sinha and Barbora (2021) has noted that the Hrusso Aka language is mostly spoken at home and primarily by those above 55 years old. The group of people who use Hrusso Aka the least are those under 25, who either do not speak the language at all or have limited proficiency.

2.1.3 Documentation and Revitalisation Efforts

When it comes to Hrusso Aka, most of these efforts are conducted by the North Eastern Institute of Language and Culture (NEILAC), which is a centre that focuses on the research, documentation and revitalisation of languages in Arunachal Pradesh (North Eastern Institute of Language and Culture, n.d.). Some of their projects include teacher training, a language academy, and publishing written materials in Hrusso Aka.

Additionally, the work of D'Souza (2015) represents the most comprehensive documentation effort of the language to date. This dataset consists of 20 hours of transcribed, translated, and glossed audio data, primarily collected in 2016 and 2017 during research fieldwork trips. This collection and documentation was part of the phonological research conducted by D'Souza (2021a).

2.2 Natural Language Processing

NLP is the interdisciplinary field of artificial intelligence and linguistics that focuses on the computational modelling and understanding of human language. The field features a wide range of applications such as machine translation, speech recognition, text normalisation, or text generation.

Due to the research scope of this thesis, the rest of this section focuses on the development of next-word prediction models, starting with N-gram models, and followed by Recurrent Neural Networks (RNNs) and transformers.

2.2.1 N-Gram Models

One of the earliest next-word prediction models relies on the intuitive observation that the presence of a word at a certain position depends on the previous words, on the context. This simple observation follows the Markov property, which states that the probability of a present state is only dependent on the past state of a system. If we extend this assumption to account for $n-1$ past states as the context for the current state, then we obtain a Markov chain of order n, or what is known in NLP as an n-gram model. In the context of n-gram next-word prediction, the Markov assumption would imply that the probability of a word to occur at position i depends only on the previous $n-1$ words that have occurred in that sentence, as seen in Equation 1.

$$P(w_i \mid w_{i-1}\, w_{i-2} \ldots w_1) = P(w_i \mid w_{i-1}\, w_{i-2} \ldots w_{i-n+1}), \text{ where } i \geq n. \tag{1}$$

The probability of a particular word, w_i, occurring is calculated according to Equation 2, where the frequency of the n-gram is divided by the frequency of the preceding $(n-1)$ words. Obtaining the next-word in a sequence becomes a maximum likelihood estimation problem, as mathematically expressed in Equation 3.

$$P(w_i \mid w_{i-1}\, w_{i-2} \ldots w_{i-n+1}) = \frac{count(w_i\, w_{i-1}\, w_{i-2} \ldots w_{i-n+1})}{count(w_{i-1}\, w_{i-2} \ldots w_{i-n+1})}, \text{ where } i \geq n. \tag{2}$$

$$\hat{w}_i = \underset{w_i}{\arg\max}\, P(w_i \mid w_{i-1}\, w_{i-2} \ldots w_{i-n+1}), \text{ where } i \geq n. \tag{3}$$

Smoothing

In the eventuality of sequences of words which were not present in the training data, smoothing is essential to overcome zero probabilities. One of the simplest techniques is Laplace smoothing. As seen in Equation 4, this method adds 1 to each count, where the V in the denominator represents the total number of n-grams (Jurafsky & Martin, 2019).

$$P(w_i \mid w_{i-1}\, w_{i-2} \ldots w_{i-n+1}) = \frac{count(w_i\, w_{i-1}\, w_{i-2} \ldots w_{i-n+1}) + 1}{count(w_{i-1}\, w_{i-2} \ldots w_{i-n+1}) + V}, \text{ where } i \geq n. \tag{4}$$

A simple variation of Laplace smoothing is Lidstone (or add-k) smoothing is assigning less of the probability mass to unseen events by adding a fractional number less than one, as defined in Equation 5 (Jurafsky & Martin, 2019).

$$P(w_i \mid w_{i-1}\, w_{i-2} \ldots w_{i-n+1}) = \frac{count(w_i\, w_{i-1}\, w_{i-2} \ldots w_{i-n+1}) + k}{count(w_{i-1}\, w_{i-2} \ldots w_{i-n+1}) + kV}, \text{ where } i \geq n. \tag{5}$$

Although simple, Laplace (or Lidstone) smoothing is not often used in practice with n-gram models. This method usually leads to poorer outcomes since it assumes that the probabilities follow a geometric distribution instead of the Zipfian distribution specific to linguistic data (Gale & Church, 1994).

Another simple approach used to overcome data scarcity is called backoff and operates on the idea that whenever we do not have examples of a particular n-gram for our inference, we could rely on fewer previous words to approximate the probability. Thus, we would keep decreasing to a lower degree n-gram until we can obtain a non-zero probability (Katz, 1987). Since we are shifting some of the probability mass to the lower degree n-grams, we need to ensure that the higher degree n-grams are discounted to avoid the total probability going above 1, which is why this method relies on having a α which controls the discounts. In practice, the backoff probability of the word w_i occurring after the context $w_{i-n+1} : w_{i-1}$ is given by Equation 6. If the n-gram was present in the training dataset, then the probability of word w_i is the discounted probability $P^*(w_i|w_{i-n+1:n-1})$. Otherwise, for an unseen n-gram, the lower degree $(n-1)$-gram is used after being weighed by $\alpha(w_{i-n+1:i-1})$.

$$P_{BO}(w_i|w_{i-n+1:i-1}) = \begin{cases} P^*(w_i|w_{i-n+1:n-1}), & \text{if } count(w_{i-n+1:i}) > 0 \\ \alpha(w_{i-n+1:i-1})P_{BO}(w_i|w_{i-n+2:i-1}), & \text{otherwise.} \end{cases} \tag{6}$$

Another variation of backoff is "stupid backoff" (Brants, Popat, Xu, Och, & Dean, 2007), which uses the relative frequencies instead of discounting. The result is no longer a probability but a score calculated based on 7, where α is the backoff factor and a unigram has score $S(w) = \frac{count(w)}{n}$. This approach resulted in a more computationally efficient approach with a similar performance as Kneser-Ney for large datasets.

$$S(w_i|w_{i-n+1:i-1}) = \begin{cases} \frac{count(w_{i-n+1:i})}{count(w_{i-n+1:i-1})} & \text{if } f(w_{i-n+1:i}) > 0 \\ \alpha S(w_i|w_{i-n+2:i-1}) & \text{otherwise.} \end{cases} \tag{7}$$

Although computationally efficient, n-gram models exhibit a clear discrepancy with natural speech, which is dynamic and not limited to often short, fixed-length sequences of words. The next subsection presents an alternative approach, which addresses the issue of long-term dependencies.

2.2.2 Recurrent Neural Networks

RNNs are artificial neural networks used to capture patterns in time series or sequences of data, where a current data point is dependent on the previous data points (Rumelhart, Hinton, & Williams, 1986). Their dynamic nature makes them a natural candidate for time series prediction applications such as forecasting the temperature, or language modelling tasks such as next-word prediction.

In previous iterations of neural networks, all inputs were processed independently from one another by maintaining strictly forward connections between the input, hidden, and output layers. In contrast, RNNs introduced cyclic connections in the hidden layer which not only require the current input but also the output of the hidden layer at the previous time step. Thus, temporal dependencies are enabled by "unfolding" the RNN through time. Similar to other neural networks, a successful RNN is one whose outputs closely follow the targets of the inputs in the training data. This similarity is quantified through a loss function, which varies depending on the specific application and type of data among others. Figure 2 represents a typical diagram for computing the loss of an RNN, where we can observe the connections in the hidden layer between subsequent time steps. In this diagram, **U**, **V**, and **W** represent the parameter matrices representing the input-to-hidden, hidden-to-hidden, and hidden-to-output connections, respectively. Our goal is to determine the optimal values for these matrices to obtain a minimal loss, i.e., a minimal difference between the network outputs and the targets.

Most RNNs are trained using an optimisation method and backpropagation through time (BPTT) (Mozer, 1995; Robinson & Fallside, 1987) to minimise the loss function. Typically RNNs are optimised through gradient descent methods by shifting the parameters such that the loss decreases, with further variations below.

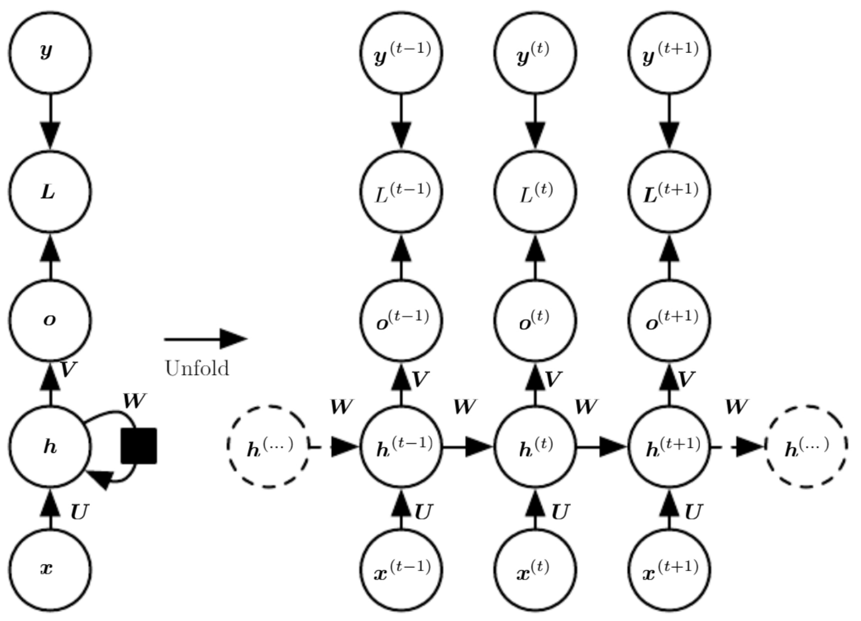


Figure 2: Typical RNN computational graph retrieved from (Goodfellow et al., 2016), where **x** are the inputs, **y** are the targets, **o** are the network outputs, and **L** is the loss function.

- **Gradient Descent.** The intuition behind gradient descent (GD) was introduced by Cauchy (1847). The motivation was to provide an alternative method for solving systems of equations which were too complicated to be solved through successive eliminations. Cauchy (1847) had the idea of representing the system as a function f of many variables $x, y, z...$ and continuously decrease the function according to its gradient until it reaches its minimum, which for a resolvable system would have the value of zero.

  Cauchy's idea later became the inspiration for gradient descent optimisation methods, as it provided an intuitive approach for minimising the objective functions during the training stages of machine learning models. Gradient descent follows Equation 8, where, at each training iteration $t+1$, the weights $\mathbf{w}_{t+1}$ are computed by shifting the weights at time t in the direction of the steepest descent provided by the gradient at time t. The degree of the weight adjustment between steps is controlled by the learning rate η_t which can be static or follow an adaptive schedule through time. One example of an adaptive approach is the Adam algorithm which is later described.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla f(\mathbf{w}_t) \tag{8}$$

- **Stochastic Gradient Descent.** In practice, a simple variant of GD called stochastic gradient descent (SGD) is often preferred. The origins of SGD can be traced back to Robbins and Monro (1951) who proposed finding the solution of an equation $M(\mathbf{x})$ by iterative estimates based on sequences of random observations of a random variable $N(\mathbf{x})$ such that $\mathbb{E}[N(\mathbf{x})] = M(\mathbf{x})$.

  In machine learning, the idea of Robbins and Monro (1951) translated into Equation 9 which computes the gradient of the objective function by using a random batch of observations i instead of using the entire dataset at each iteration.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla f_i(\mathbf{w}_t) \tag{9}$$

- **Adam.** One of the most popular optimisation approaches was introduced by Kingma and Ba (2015) as the successor of Adaptive Gradient (AdaGrad) and Root Mean Square Propagation (RMSProp) optimisation. AdaGrad tunes a different learning rate for each parameter using the sum of its gradients, acknowledging how convergence might benefit from updating the parameters with different frequencies (Duchi, Hazan, & Singer, 2011). RMSProp uses a similar adaptive approach by using a decaying gradient average which leads to better convergence by allowing the model to "focus" on gradients seen more recently during training (Tieleman & Hinton, 2012). Finally, Adam uses two averages to adapt the learning rate of each parameter. Namely, Adam uses an average based on the gradients, and the other based on the gradients squared, which was proved to work well on sparse gradients and gradually reduce learning rates which allow the parameter updates to "slow down" around local minima (Kingma & Ba, 2015).

However, one needs to calculate the gradients during optimisation, for which BTTP is employed. BPTT uses the "unfolded" network, taking into account the current and previous states of the network, and relies on the same idea as generalised backpropagation. Namely, BPTT iteratively minimises the loss function based on its gradients with respect to the network parameters. As the name states, the gradients are propagated backwards through time steps using the chain rule. These repeated gradient multiplications give rise to two main challenges: *(i)* vanishing gradients, where the gradients approach zero and cause slow long-term dependency learning, and *(ii)* exploding gradients, where gradients become large and lead to unstable learning (Goodfellow et al., 2016).

RNN Variations
The Long Short-Term Memory (LSTM) model is a variation of the RNN designed to capture long-term dependencies while overcoming vanishing and exploding gradients (Hochreiter & Schmidhuber, 1997). This is achieved by introducing a series of three gates - forget, input, and output - that control how the input at the current time step, x_t, and the previous hidden state, h_{t-1}, contribute towards the current hidden state, h_t. Specifically, the forget gate, f_t, learns what information from the previous time step to be discarded. The input gate, i_t, determines the information to be stored. Lastly, the output gate, o_t determines how much of the cell state is to be passed in the next time step. The LSTM unit can be visualised in Figure 3 and is defined by Equations 10-15, representing the gates, cell state update, and hidden state vector. In these equations, *(i)* σ represents the sigmoid function, whose range, $(0,1)$, allows the outputs of the gates to be interpreted as probabilities, *(ii) tanh* represents the hyperbolic tangent function used to mitigate against vanishing gradients (Hochreiter & Schmidhuber, 1997), *(iii)* W,U per each gate and the cell state represent the weight matrices *(iv)* b_f, b_i, b_o, b_C are the corresponding bias vectors, *(v)* x_t is the input at time t, *(vi)* h_{t-1} and h_t are the previous and current hidden states, respectively, and *(vii)* C_{t-1} and C_t are the previous and current cell states, respectively.

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{10}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{11}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{12}$$

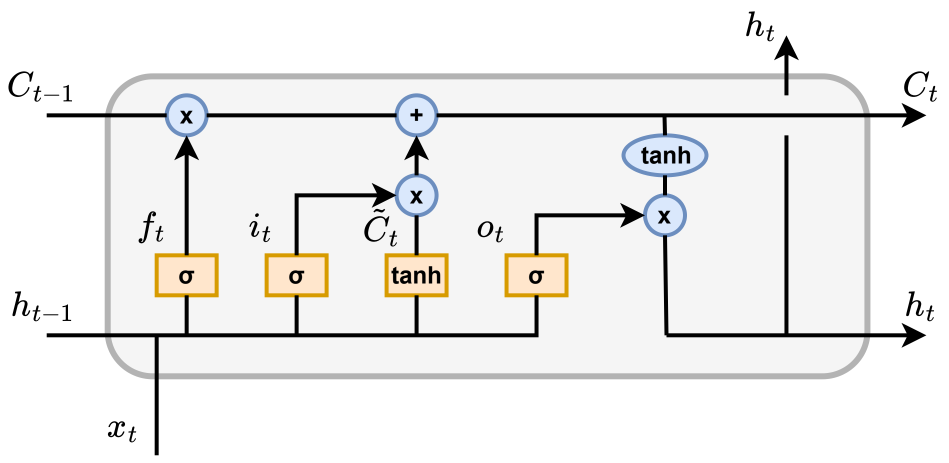


Figure 3: LSTM unit diagram.

$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{13}$$

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t, \text{ where } \circ \text{ is the element-wise multiplication} \tag{14}$$

$$h_t = o_t \circ tanh(C_t) \tag{15}$$

Since LSTMs can often be computationally expensive due to the large number of parameters, simpler gated RNNs have been proposed. One solution is the Gated Recurrent Unit (GRU) model which uses a single gate to replace the forget and update units (Cho, Van Merriënboer, Gulcehre, et al., 2014). Figure 4 illustrates the architecture of the GRU unit, which is defined by a reset gate and an update gate. The two gates and the hidden state are defined by Equations 16, 17, and 18. In these equations, *(i)* σ represents the sigmoid function, *(ii)* *tanh* represents the hyperbolic tangent function, *(iii)* W_r, W_z, W, U_r, U_z, U are the weight matrices, *(iv)* x_t is the input at time t, and *(v)* h_{t-1} and h_t are the previous and current hidden states, respectively.

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \tag{16}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \tag{17}$$

$$h_t = z_t \cdot h_{t-1} + (1 - z_t) \cdot \tilde{h}_t \tag{18}$$

$$\tilde{h}_t = tanh(W x_t + U(r_t \circ ht - 1)) \tag{19}$$

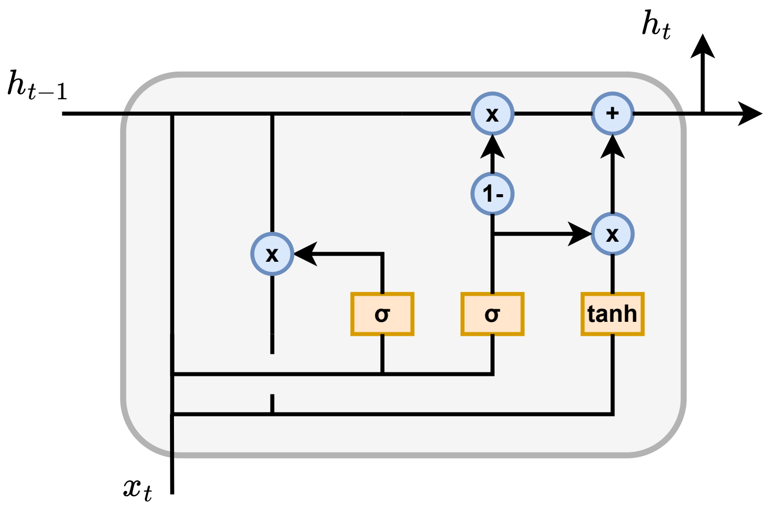


Figure 4: GRU unit diagram.

2.2.3 Transformer Models

The current state-of-the-art models in many NLP tasks, including next-word prediction, are the transformer-based models, which were first introduced by Vaswani et al. (2017). However, before discussing the transformer architecture, it is essential to understand its evolution from the earlier sequence-to-sequence (*seq2seq*) models and the attention mechanism.

Sequence-to-Sequence Models

The *seq2seq* model was proposed by Sutskever, Vinyals, and Le (2014) in a machine translation task. This method aimed to provide an end-to-end approach to sequence generation without strict assumptions over the sequence structure (e.g. sequence length). The *seq2seq* model used two LSTM networks to achieve its goal: one serving as an encoder and the other as a decoder. The encoder processed the input sentence, into a fixed-length context vector while the decoder mapped the resulting context vector to the target sentence. The architecture proposed by Sutskever et al. (2014) proved to be an improvement compared to previous statistical machine translation approaches even with limited optimisation, while correctly translating even very long sentences by creating many short term dependencies solely through reversing the word order.

Nonetheless, a limitation of the *seq2seq* models is that the performance often drops as the input sentences become longer (Cho, Van Merriënboer, Bahdanau, & Bengio, 2014). Cho, Van Merriënboer, Bahdanau, and Bengio (2014) chose 620-dimensional context embeddings and hypothesised that the network might have “sacrificed” important parts of long sentences due to limited capacity in the embeddings.

The Attention Mechanism

The attention mechanism is another encoder-decoder approach proposed by Bahdanau, Cho, and Bengio (2014) to address the limitations highlighted by Cho, Van Merriënboer, Bahdanau, and Bengio (2014). Instead of representing sentences using fixed-length vectors, Bahdanau et al. (2014) proposed encoding the sentences into sequences of vectors and adaptively relying on a subset of encoding vectors.

Thus, the context vector was presented a weighted sum of encoded words $h_1, ..., h_{T_x}$, where T_x is the length of the input sentence, as seen in Equation 20.

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{20}$$

The weights are computed according to Equation 21, where e_{ij} (Equation 22) is a feedforward neural network (FFN) which learns the alignment between position j of the input and position i of the output, based on the previous hidden state s_{i-1}. The alignment model is trained simultaneously with the encoder and decoder, resulting in an architecture which is able to focus on relevant parts of a sentence regardless even for longer, complex sentences.

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T^x} \exp(e_{ik})} \tag{21}$$

$$e_{ij} = a(s_{i-1}, h_j) \tag{22}$$

The Transformer Architecture

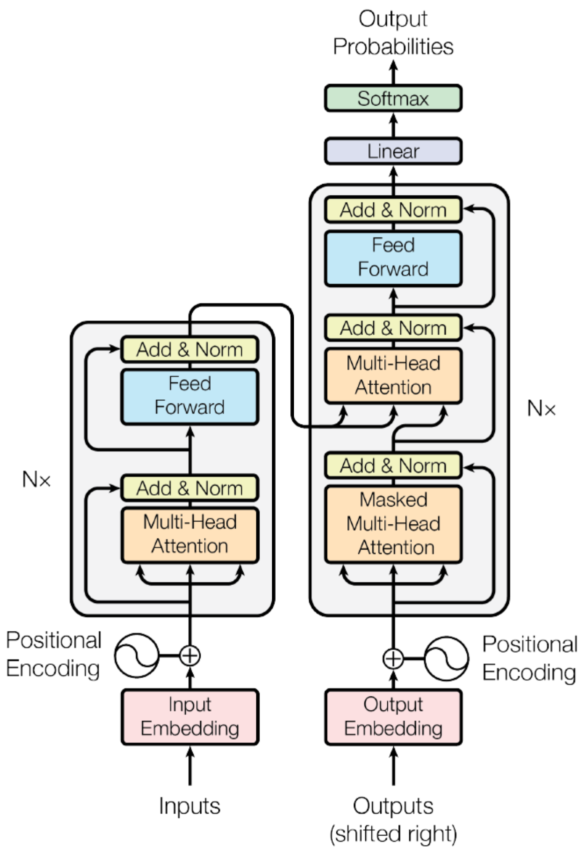


Figure 5: Transformer architecture diagram. Image retrieved from (Vaswani et al., 2017).

The transformer architecture was proposed by Vaswani et al. (2017) building onto the knowledge of *seq2seq* models and attention mechanisms. The authors proposed a simpler architecture that uses only attention mechanisms in both the encoder and the decoder, discarding the computationally expensive RNN models.

The architecture is presented in Figure 5. Given an input sequence $\mathbf{x} = (x_1, ..., x_n)$, the encoder maps it to a new vector $\mathbf{z} = (z_1, ..., z_n)$, which is then mapped to the output sequence $\mathbf{y} = (y_1, ..., y_m)$.

The encoder is a stack of identical layers, where each layer consists of a multi-head self-attention mechanism and a fully connected FFN. Around each of the layers, residual connections and layer normalisation are used.

The decoder follows the architecture of the encoder except for an additional multi-head attention layer over the encoder's output. To ensure that the predictions at position i only depend on preceding outputs, the self-attention layer of the decoder is also modified such that all succeeding positions are masked.

The attention function proposed with the transformer model is the scaled dot-product attention expressed in Equation 23, where Q is the query, K is the key, V is the value, and d_k is the dimension of the queries, and the keys. Although this type of attention performs similarly to the FFN mechanism proposed in (Cho, Van Merriënboer, Gulcehre, et al., 2014), it is much faster and space-efficient.

$$\text{Attention}(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{23}$$

The attention mechanism is then concatenated into multi-head attention as defined in Equation 24, where W_i^Q, W_i^K, W_i^V, and W_i^O are the parameter matrices at position i of the query, key, value, and output, respectively.

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O, \text{ where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{24}$$

Lastly, to capture the order of the sequences, positional encodings are added to the input and output embeddings of length d_{model}, which are simply sine and cosine functions of varying frequency.

Variations

The default transformer architecture can also be further customised depending on the task. The main variations can be classified into encoder-only or decoder-only approaches.

Encoder-only models such as the Bidirectional Encoder Representations from Transformers (BERT) model achieved state-of-the-art performance in sentence classification, language inference, and even question-answering (Devlin et al., 2018). The base model uses 12 encoder layers, no decoder layers, a hidden size of 768, and 12 attention heads.

In next-word prediction though, decoder-based models such as Generative Pre-Training (GPT) have had the most important impact (Radford et al., 2018). The model uses no encoder layers, 12 decoder layers, a hidden size of 768, and 12 attention heads.

2.3 (Very) Low-Resource Language Representation in NLP

Although the methods discussed in Section 2.2 have been employed in numerous tasks and for multiple languages, the current section provides an outline of the most relevant publications concerning the task of next-word prediction in very low-resource languages with corpora of less than 1 million tokens.

2.3.1 N-grams

Hamarashid, Saeed, and Rashid (2021) developed an N-gram model for next-word prediction in the Sorani and Kurmanji dialects of Kurdish. The study relied on a corpus of 505 000 words and compared the efficiency of bigrams, trigrams, quadgrams and pentagrams. To overcome the sparsity issue of small datasets, the authors used the stupid back-off algorithm, which decreases the number of words in the *n*-gram when there is insufficient evidence for a prediction and uses interpolation to combine all the *n*-grams. Since the model was integrated into software, the model provides the user with the 5 most likely words. The accuracy was calculated by assessing whether the target word was part of the five most likely predicted words. The authors found that the accuracy of the model increased with the number of words in the *n*-gram, where the best accuracy was 96.3% for the pentagram model.

Another study is that of Balouch (2024), who used an N-gram model for next-word prediction in the Balochi language, a potentially endangered language from Pakistan (UNESCO, 2024). Although the Balochi language is spoken by approximately 10 million people, the corpus used by Balouch (2024) consisted of only 33 218 sentences, totalling 317 590 words. The study compared bigrams, trigrams, quadgrams, and pentagrams. To handle data sparsity, the author compared Laplace and Lidstone smoothing. The models were extrinsically evaluated based on accuracy, and intrinsically evaluated based on perplexity. The models were intrinsically evaluated based on perplexity, whose and extrinsically evaluated based on accuracy. Overall, Lidstone smoothing resulted in models with lower perplexity compared to Laplace smoothing. Additionally, as the number of words in the N-gram increased, the perplexity decreased and the accuracy increased. Consequently, the best model was the Lidstone smoothing pentagram model which achieved an accuracy of 93% and a perplexity of 240.

2.3.2 Recurrent Neural Networks

Rakib, Akter, Khan, Das, and Habibullah (2019) developed a GRU-based RNN model for Bangla next-word prediction on a dataset of only 170 000 words. In their study, they trained 5 models for different input lengths: *(i)* unigram GRU RNN, *(ii)* bigram GRU RNN, *(iii)* trigram GRU RNN, *(iv)* quadgram GRU RNN, and *(v)* pentagram GRU RNN. All 5 models used the same architecture, namely, an embedding layer of dimensionality 50, two GRU units with 100 neurons each, one dense layer with 100 neurons, and a final dense layer for the output. After being trained for 1000 epochs the accuracies were: *(i)* 32.77% for the unigram model, *(ii)* 78.15%, *(iii)* 95.84%, *(iv)* 99.24% for the quadgram model, and *(v)* 99.70% for the pentagram model. However, this study suffers from reproducibility issues. No information is provided regarding the optimisation algorithm. Another factor that hinders the reproducibility of this study and questions the validity of the results is the lack of information concerning how data was handled during training and testing. Since the authors have not specified whether the accuracies provided were obtained during training or testing, it is possible that the entire dataset was used during training. This would imply an unrealistically high accuracy due to possible overfitting.

2.3.3 Transformer Models

A literature search revealed no publications targeting transformer models developed specifically for next-word prediction in low-resource languages. Nonetheless, a notable trend has emerged amongst transformers developed for other tasks in low-resource languages: compact models tend to perform better. The following paragraphs illustrate this finding by describing transformers developed for neural machine translation (NMT) and text normalisation.

Neural Machine Translation

NMT in low-resource languages is likely the most commonly addressed NLP task using transformers.

For example, Van Biljon, Pretorius, and Kreutzer (2020) compared shallow (1 encoder and 1 decoder), medium (3 encoders and 3 decoders), and deep (6 encoders and 6 decoders) transformers in English to Setswana, Sepedi, and Afrikaans tasks. The shallow and medium transformers outperformed the deeper architecture on the three datasets ranging from 30 777 sentences to 123 868 sentences.

Additionally, Araabi and Monz (2020) evaluated a *seq2seq* model on parallel datasets of English and Belarusian, Galician, Slovak, or Slovenian. The 4 datasets ranged in size from 4 500 to 55 000 sentences. The study has found that smaller datasets required shallower and narrower transformers with a higher dropout rate.

Similarly, Lankford, Afli, and Way (2024) found that decreasing the number of hidden neurons and increasing the dropout rate improved the performance of NMT on English-Irish parallel data. The transformer model outperformed a baseline RNN model and was trained on a general corpus of 55 900 lines and an in-domain corpus of 91 500 lines.

Text normalisation

Lusito, Ferrante, and Maillard (2023) examined the performance of a *seq2seq* model in a Ligurian text normalisation task. Ligurian is an endangered language (UNESCO, 2024), resulting in four limited datasets, totalling 4 394 sentences and their normalised versions. Their architecture consisted of 4 encoder and decoder layers, 4 attention heads, word embeddings of dimensionality 256, 1024 neurons in the hidden layers, dropout of 0.3, and label smoothing of 0.1. During training an inverse root scheduler was used alongside Adam, where the learning rate was 10^-3, the batch size was 20, and the number of warmup updates was 4000. The model was trained once per dataset, followed by training on the union of the datasets. The joint model led to better text normalisation and outperformed the models trained per dataset. The character error rate of the joint model was below 5%, which the authors considered a success and concluded that compact transformers could be useful tools for language preservation.

2.4 Fine-Tuning Machine Learning Models

As can be observed from the previous subsections, training a machine learning model is a complex process, which involves the consideration of various factors such as, amongst others, the number of layers, the dimensionality of the word embeddings, or the choice of regularisation and optimisation strategies. Given the large number of possible configurations, it is more than advisable to guide our choices sensibly based on previous research. This approach would allow us to identify promising baseline configurations, and to become aware of possible challenges.

Although it is possible to perform a comprehensive grid search over a hyperparameter search space defined based on previous research and personal intuition, in some cases, related literature might be limited to some machine learning applications. In such situations, we could consider automatic hyperparameter optimisation frameworks such as Optuna (Akiba, Sano, Yanase, Ohta, & Koyama, 2019), which implements a variety of search algorithms and enables parallelisation.

Some of the hyperparameter optimisation algorithms that Optuna implements are: *(i)* grid search, suggesting all possible combinations in the search space; *(ii)* random sampler, which performs independent sampling in the search space; *(iii)* Tree-structured Parzen Estimator (TPE), a Bayesian optimisation method that prioritises choosing hyperparameters from areas of that search space that has led to better model performance (Bergstra, Bardenet, Bengio, & Kégl, 2011); or *(iv)* Nondominated Sorting Genetic Algorithm II (NSGA-II), a type of genetic algorithm that preserves the best configurations of the current generation in the future generation while prioritising population diversity within the best-performing individuals (Deb, Pratap, Agarwal, & Meyarivan, 2002).

Additionally, Optuna offers a hyperparameter importance functionality, meant to inform the user which hyperparameters have contributed the most to the performance of the trained model. Optuna implements functional ANOVA (fANOVA), which uses random forests to determine how much performance variance is determined by a particular hyperparameter or combinations of hyperparameters (Hutter, Hoos, & Leyton-Brown, 2014).

2.5 Federated Learning

Another possible solution to improve the performance of a model is to fine-tune it on new data. This approach can be utilised regardless of whether the hyperparameters of the original model have been previously fine-tuned or whether the model made use of a tried and true hyperparameter configuration. Fine-tuning a model on additional data is especially suitable when the initial model was trained on scarce data when the predictions made by the model are less likely to be generalisable. An intuitive approach for a model that has been deployed and is actively used would be collecting data from the users, sending it to a server, updating the model and sending the updated model back to the users. Although simple, this method suffers from a clear data privacy vulnerability by allowing user data to be shared outside their device.

An alternative was proposed by Konečný, McMahan, and Ramage (2015), who introduced a distributed optimisation technique known as federated learning. All distributed optimisation techniques have the purpose of minimising an objective function f similarly to 25 while splitting the data across multiple nodes and performing the computations in parallel (Konečný et al., 2015).

$$\min_{w\in\mathbb{R}^d} \quad \text{where} \quad f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w) \tag{25}$$

Traditional distributed systems traditionally faced two main limitations. Firstly, they encountered communication bottlenecks if communication between the main server and the nodes was performed at each iteration of the optimisation algorithm (Balcan, Blum, Fine, & Mansour, 2012). Secondly, data often needed to meet restrictive assumptions, such as having the same amount of samples in each node or the data in each node being a good representation of the overall data distribution (Shamir, Srebro, & Zhang, 2014).

Algorithm 1 Federated learning algorithm. Retrieved from (Konečný et al., 2016).

```
1: parameters: $\eta$ = stepsize, data partition $\{P_k\}_{k=1}^{K}$, diagonal matrices $A, S_k \in \mathbb{R}^{d\times d}$ for $k \in \{1,\dots,K\}$
2: for $s = 0,1,2,\dots$ do                                          ▷ Overall iterations
3:     Compute $\nabla f(w^t) = \frac{1}{n}\sum_{i=1}^{n}\nabla f_i(w^t)$
4:     for $k = 1$ to $K$ do in parallel over nodes $k$               ▷ Distributed loop
5:         Initialize: $w_k = w^t$ and $h_k = h/n_k$
6:         Let $\{i_t\}_{t=1}^{n_k}$ be random permutation of $P_k$
7:         for $t = 1,\dots,n_k$ do                                  ▷ Actual update loop
8:             $w_k = w_k - \eta_k(S_k[\nabla f_{i_t}(w_k) - \nabla f_{i_t}(w^t)] + \nabla f(w^t))$
9:         end for
10:    end for
11:    $w^t = w^t + A\sum_{k=1}^{K}\frac{n_k}{n}(w_k - w^t)$                  ▷ Aggregate
12: end for
```

In comparison, the method proposed by Konečný et al. (2016) is well-suited for sparse data by handling nodes with varying numbers of samples that may be representative of different distributions. In Algorithm 1, the global model w^t is updated by aggregating the local updates w originating from K nodes proportionally to the partition size n_k. For each node k, the local model w_k is initialised with the global model w^t, then updated by incorporating local and global gradient information modulated by a local stepsize $\eta_k = \frac{\eta}{n_k}$. Lastly, the two matrices S_k and A allow the model to adapt to non-independent and identically distributed data by scaling the updates according to the globally observed distribution.

Since its proposal, federated learning has been employed successfully in a variety of tasks such as binary classification (Konečný et al., 2015), electroencephalography prediction (Gao et al., 2019), or next-word prediction (Hard et al., 2018), which will be described in detail further.

2.5.1 Applications in Next-Word Prediction

In 2018, Google implemented an on-device federated learning framework for mobile next-word prediction (Hard et al., 2018). Given the use-case of the model, Hard et al. (2018) opted for Coupled Input-Forget Gate (CIFG) LSTM, which resulted in less parameters and a more compact model by using only one gate to control the input and the forget gates. Additionally, the architecture features only one hidden layer of 670 units, and a word embedding dimensionality of 96.

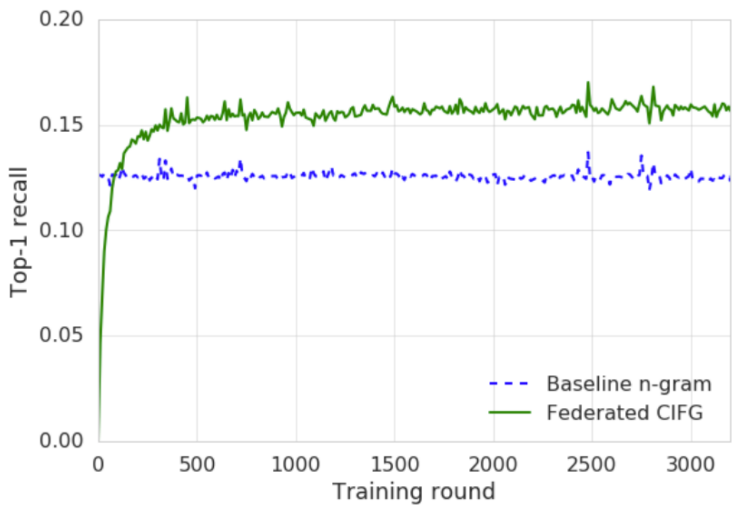


Figure 6: Top-1 recall as a function of federated training round. Image retrieved from (Hard et al., 2018).

In the study, 7.5 billion sentences were used for training and 25 000 sentences were used for testing, with an overall sentence length of 4.1 words. The data came mostly from chat applications (60%), followed by web input (35%), and long form text (5%). The federated-updated CIFG was compared to server-updated CIFG, and with an *n*-gram baseline model. Overall, both CIFG models performed better than the *n*-gram model, obtaining a top-1 and top-3 recall of approximatively 16.5% and 27%, respectively, while the *n*-gram model's top-1 recall was 13% and its top-3 recall was 22.1%. The federated learning approach converged after 3000 training rounds, which can be observed in Figure 6 alongside the performance of the n-gram model, which converged quicker but had a lower top-1 recall. Additionally, in the server-based setting, the authors compared the performances of Adam, AdaGrad and SGD, which showed that the adaptive gradient methods did not improve convergence compared to SGD.

2.6 Mobile Application Design

Developing a high-quality next-word prediction keyboard goes beyond quantitative measures such as prediction model accuracy, application responsiveness, or memory requirements. The rest of this subsection is dedicated towards exploring qualitative aspects such as user interface and user experience in relation to existing popular keyboard applications.

2.6.1 User-Centered Design

The term user-centered design was introduced and popularised in the late 1980s. Norman (1986) recognised the need of thoroughly considering the needs of the users when designing the interface, since as far as they are concerned "the interface is the system" (Norman, 1986, p. 61).

Since then, multiple authors such as Shneiderman (1987), Norman (1988), or Cooper (1995) have proposed their own sets of user-centered design principles contributing towards user-friendly and error-tolerant applications. The published guidelines emphasised amongst others: *(i)* reducing the cognitive load by limiting distractions through a minimalist interface that allows the user to maintain control over the task, *(ii)* designing for possible errors and allowing for information retrieval, and *(iii)* ensuring that the mappings between the buttons and their result are consistent and intuitive, guiding the user towards the desired action.

The principles proposed in the 1980s and 1990s laid the foundation for the evolution of the field, which later stressed the active involvement of users in the design process (Abras, Maloney-Krichmar, Preece, et al., 2004). For example, Preece, Rogers, and Sharp (2002) proposed involving the users from the incipient stages of designing the product until the final stage of the process. In the beginning stages, the users could provide data relating to their needs and expectations through questionnaires and focus groups. Later on, the users could take part in product simulations, user satisfaction questionnaires, or usability testing such as measuring how long it takes a user to learn how to perform a task or assessing the number and type of errors that are most prevalent.

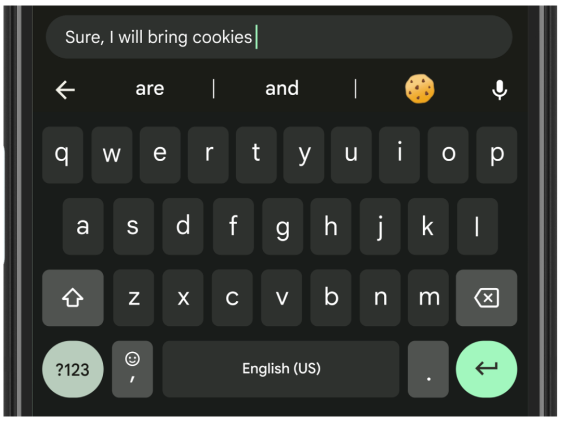
The area of user-centered design is vast and could be further described. However, in the context of developing mobile keyboards, the design process is simpler due to the many already existing applications. Therefore, understanding the current practices, which will be further discussed, is important to minimise the adjustment time from one framework to another.

2.6.2 Keyboard Examples

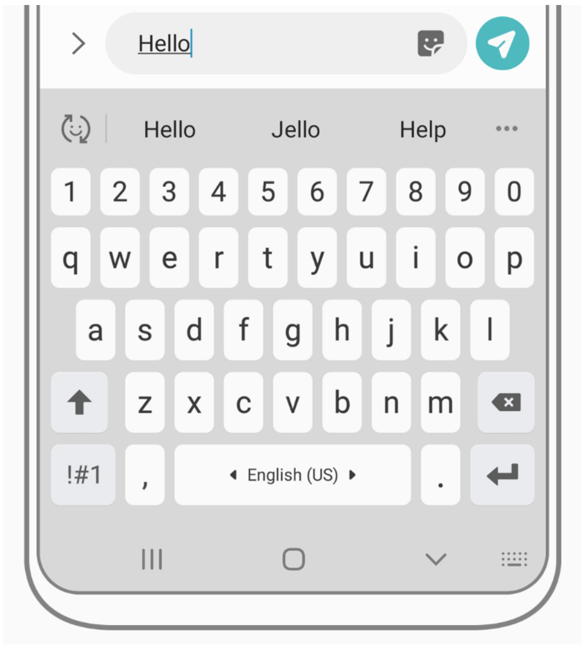
The leading mobile keyboard applications of today feature a diverse array of additional functionalities besides just typing. Two of the most widely used mobile keyboards are the Google keyboard (Google LLC, n.d.) and the Samsung Keyboard (Samsung Electronics Co., Ltd., n.d.), both of which offer advanced tools such as next-word prediction, voice-to-text, emoji support, or even built-in translation capabilities. Although these keyboards offer such a multitude of complex functionalities, their interfaces remain intuitive.

In Figure 7, we can observe that the Google and Samsung keyboards have a clean and minimalist design with well-spaced keys and highly contrasting colour palettes. Both keyboards adhere to the QWERTY arrangement and action keys, such as “Return” and “Backspace”, are found in the standard positions providing predictability across frameworks. The typing flow is maintained by placing the buttons for additional features, such as accessing special symbols or activating text-to-speech, outside of the user’s main focal area (i.e. the keys and the input field), which ensures that distractions are minimal. Moreover, the now ubiquitous next-word prediction bar is conventionally placed between the input field and the main keys. This location further supports the typing flow by enabling users to view and select the desired word without shifting their attention away from the main focal area.

Since Hrusso Aka makes use of diacritics, careful consideration was given to an efficient approach, which is presented in detail in Section 3.5. However, the approach that the two keyboards in Figure 7 use is allowing users to access less frequent characters, such as variations caused by diacritics, in a separate pop-up window without cluttering the view. This is a feature shared by both mainstream applications described above, which is accessed through a long press on the letter to access its diacritics.



(a) The Google keyboard. Image retrieved from (Google LLC, n.d.).



(b) The Samsung keyboard. Image retrieved from (Samsung Electronics Co., Ltd., n.d.).

Figure 7: Visual comparison of the user interface of two widely used mobile keyboards.

3 Methods

This section presents the methodology used in this thesis. The section commences with a detailed overview of the dataset utilised along with the methods used to preprocess it. A description of the models and their architecture follows, alongside the methodologies for hyperparameter tuning and model evaluation. Finally, this section provides an overview of the mobile keyboard design decisions, their motivation, and some specifications of the keyboard implementation process.

3.1 Dataset

To reiterate, in this subsection, the dataset, its size and its conversation topic distribution are discussed. The procedure used during preprocessing and the specifics of splitting the dataset follow.

3.1.1 Overview

The dataset consists of 20 hours of Hrusso Aka audio data transcribed, translated into English, and glossed using ELAN (*ELAN [Computer software]*, n.d.). The audio has been primarily collected in 2016 and 2017 (D'Souza, 2021b). This dataset is a broad representation of the Hrusso Aka language since it includes a mix of 70% natural language, such as conversations and narratives, and 30% elicited or semi-elicited data (i.e. artificial data). Moreover, it covers various genres, including conversations, storytelling, oral history, procedural texts, folklore, mythology, child speech, shamanic chanting, riddles, or proverbs.

As an early observation into the suitability of the dataset for a next-word prediction keyboard, we can note that the dataset was gathered from speech and not mobile texts. Additionally, our target group is the youth, who would most likely engage in topics that are different compared to the genres addressed by the current dataset. These two limitations are discussed in detail in Section 5.1.1.

To further assess the diversity of the topics covered in the dataset, we extracted one keyword for each sentence translated into English using a BERT-based model [1]. The most frequent 50 keywords were then plotted in Figure 8, where we can observe some recurring themes such as traditional occupations, family life, or local flora and fauna.

Additionally, the dataset is fully normalised, where none of the sentences contain any punctuation apart from hyphens in compound words, and where the only capitalised words are the named entities. An example data point is given in the example below.

(1) Fadar Vijay du
Father Vijay exist
"(since) Father Vijay is here."

In total, the dataset comprises 11 464 sentences with an average length of 7 tokens and a maximum length of 54 tokens. The total number of tokens in the dataset is 84 854, of which 16 062 are unique

[1] Maarten Grootendorst, *KeyBERT: Minimal Keyword Extraction with BERT*, accessed August 10, 2024, `https://maartengr.github.io/KeyBERT/`.

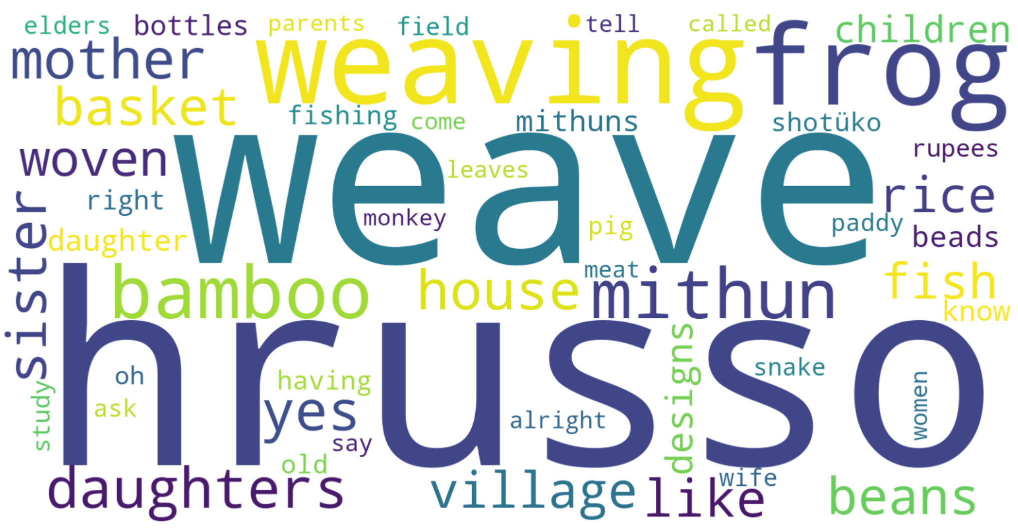


Figure 8: Most common words in the dataset.

words.

3.1.2 Preprocessing

The preprocessing steps require to be distinguished based on scope, which can be model development or inference during keyboard usage. Firstly, the data used for model training and evaluation described in Section 3.1.1 requires minimal preprocessing since it has already been processed by hand before being made available. In contrast, the raw data generated by users while interacting with the keyboard must undergo additional preprocessing steps described in detail below.

Training and Evaluation Data Preprocessing
The only preprocessing steps applied to the dataset used for model training and evaluation are adjusting from spoken to written language characteristics, as in the two steps below.

1. Because the data was adapted from audio recordings, several markers were used to indicate laughter, fillers and indistinct speech. Thus, all occurrences of "FILLER", "filler", "LAUGH", and "INDISTINCT" were removed from all sentences.

2. All word duplicates occurring consecutively were removed. After examining the dataset, word reduplication appeared to be an artefact of spoken language and not a linguistic phenomenon serving a syntactic purpose.

User-Generated Data Preprocessing

User-generated data undergoes the simple preprocessing pipeline illustrated in Figure 9. This pipeline has two goals: *(i)* it ensures that, at inference time, the input aligns with the training data, and *(ii)* it guarantees that any data stored for federated learning updates can provide insights into additional named entities. Each preprocessing step is described below.

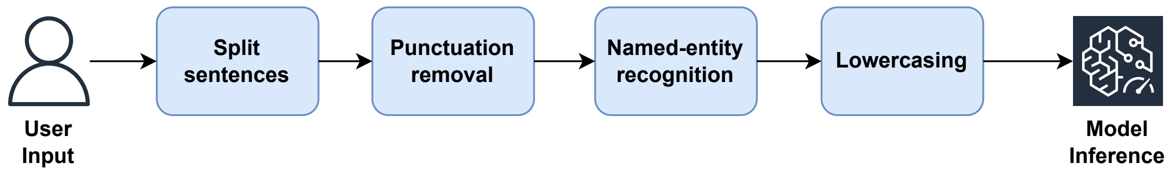


Figure 9: User-generated data preprocessing pipeline.

1. Split sentences: The user's input is split into sentences based on end marks such as periods, question marks, and exclamation points. If punctuation is missing, which might happen while using a messaging application, the entire input will be treated as one sentence and the predictions might be of lower quality due to using a potentially out-of-context word during inference time.

2. Remove punctuation: All punctuation marks are removed apart from inner-word hyphens which preserve the meaning of compound words.

3. Named entity recognition: Although the training data contained capitalised named entities, these instances are a small subset when putting into perspective the small amount of data we had available. For inference purposes, the prediction based on unknown words will not be affected regardless of whether we consider them a named entity or not. However, if we were to update the models locally based on user-generated data, we could consider all capitalised words that are not located at the beginning of a sentence named entities. Ideally, a dedicated named entity recognition system could be deployed. Since none is available, our best strategy would be trusting the users with their input, which is not ideal since many people do not adhere to writing standards when using their phones.

4. Lowercasing: All words at the beginning of a sentence are converted to lowercase unless they have been identified as named entities. Any other capitalised words entered by the user will be treated as named entities and will remain unchanged.

3.1.3 Data Splitting

The dataset was partitioned into three random subsets: training, validation, and testing. For training the models, 80% of the data was allocated. For validation during hyperparameter tuning, 10% of the data was allocated. The remaining 10% of the data was allocated towards testing the performance of the optimal models determined after optimisation.

3.2 Models

This subsection discusses the three families of NLP models selected for implementation, ordered by model complexity — starting with n-gram models, followed by RNN models, and concluding with transformer models.

3.2.1 N-Grams

The *n*-gram model is the least complex compared to the other two architectures and will serve as a baseline for evaluating the other models.

To compare the effects of smoothing in the task of next-word prediction for very low-resource languages, four variations of the model are implemented: *(i) n*-grams with no smoothing, *(ii) n*-grams with Laplace smoothing, *(iii) n*-grams with Lidstone smoothing, and *(iv) n*-grams with StupidBackoffSmoothing. Afterwards, each of the models was compared by varying the number of context words from one to five.

3.2.2 RNNs

Each input sentence is first tokenised using the spacing between the words as a separator. The sentence is then converted from a sequence of words to a sequence of integers, where each unique word in the training dataset was previously assigned a unique integer identifier. Before feeding the dataset to the neural network itself, we need to ensure that all sequences are of equal length and pad the shorter sentences to the left accordingly. We decided on vectors of dimensionality 54, the maximum sentence length encountered in the training data split.

The proposed RNN model, implemented using TensorFlow's Keras library, is a sequential neural network consisting of the following three layers:

1. **Embedding Layer:** This layer maps each word from the sequence into a dense vector. These embeddings are learned during training, allowing the model to assign more similar vectors to semantically related words.
2. **RNN Layer:** The implemented models contained only one hidden layer, similarly to Hard et al. (2018), namely, either an LSTM layer or a GRU to explore whether more compact layers such as the GRU would lead to a comparable or better performance as the more complex LSTM. Although Hard et al. (2018) used a CIFG layer to obtain a compact model, the GRU layer was preferred due to being more highly available in existing Python frameworks, as well as more common in literature, proving its reliability in a variety of tasks.
3. **Dense Layer:** A dense layer using the softmax activation function was added to the output layer. This function converts the output of the hidden layer to a probability distribution over the vocabulary, giving an indication of which words are most likely to occur next in the sequence.

The hyperparameters tuned are further described in Section 3.3.2.

3.2.3 Transformer

Since our goal is text generation, a decoder-only architecture such as that of GPT (Radford et al., 2018) is appropriate.

Similar to the RNN model, each input sequence is first tokenised into words using the spacing between words as a separator. Each word is assigned a unique identifier and the sentences are converted from sequences of words to sequences of integers.

The proposed architecture, implemented using the *fairseq* toolkit, is composed of the following blocks:

1. **Embedding Layer:** The layer maps the input tokens of to dense vectors of any desired dimensionality.
2. **Positional Encoding:** Positional encodings are added to the embedding vectors, where both have the same dimensionality.
3. **Decoder Layer:** Which follows the structure described in Section 2.2.3, *(ii)* a masked multi-head self-attention layer to capture the dependencies between the current and preceding words; *(ii)* a fully connected FFN.
4. **Output Layer:** The output is obtained using a linear layer followed by softmax activation. Similarly to the RNN models, this converts the output to a probability distribution over the vocabulary for the next token prediction.

The architecture was then fine-tuned according to the hyperparameter configurations in Section 3.3.3.

3.3 Hyperparameter Tuning

This subsection details the hyperparameter tuning schedule of the three models in the following order: *n*-gram, RNN, and transformer model.

3.3.1 N-Gram Model

To optimise model performance, hyperparameter tuning was performed by varying the number of context words used for prediction between one and five. These values are similar to previous *n*-gram studies (Hamarashid et al., 2021; Balouch, 2024) and are plausible given the average sentence length of 7 tokens in our dataset. Furthermore, based on the dataset collected by Hard et al. (2018) from Google keyboard users, where the average sentence length was 4.1 words, we do not have any reason to expect long sentences from Hrusso Aka keyboard users.

In addition to adjusting the context window, the smoothing parameter k from Lidstone smoothing was set to 0.5 similar to (Balouch, 2024). For StupidBackoff, the backoff factor α was set at 0.4 as set by Brants et al. (2007) in the original proposal of StupidBackoff.

3.3.2 RNN Model

To optimise the RNN model, the hyperparameters were varied according to Table 1. The optimisation was carried out using the automatic hyperparameter optimisation framework Optuna (Akiba et al., 2019) described in Section 2.4.

Although not completely sound in terms of reproducibility, we took some inspiration from the next-word prediction model of Rakib et al. (2019), who used a GRU model with an embedding dimension of 50. As further inspiration, we looked at the next-word prediction model of Hard et al. (2018), who used a batch size of 50, an embedding dimension of 96, one hidden layer of 670 units, and experimented with SGD and Adam. Thus, we tried keeping the search space in the same range as Rakib et al. (2019) and Hard et al. (2018). Additionally, we wanted to compare the performance of a more complex model such as the LSTM alongside the GRU model. The learning rate was set to either 10^{-3}, the TensorFlow default, or to 10^{-4} to compare whether slower convergence would be beneficial in a low-resource setting. Finally, the patience was set semi-arbitrarily, taking into account that such a small dataset would intuitively not take many epochs to converge.

In total, 50 Optuna experiments were conducted using the TPE algorithm, which offers the advantage that during optimisation the sampler prioritises configurations that are more similar to the better-performing options from the past. However, it was chosen arbitrarily in favour of NSGA-II, which also takes advantage of the better configurations from the past.

Hyperparameter	**Values**
Embedding dimension	50, 100
RNN type	LSTM, GRU
Number of hidden layers	1
Number of units in hidden layer	128, 256, 512
Optimiser	SGD, Adam
Learning rate	10^{-3}, 10^{-4}
Batch size	16, 32, 64
Early stopping patience	5

Table 1: Hyperparameter values for RNN model optimisation.

3.3.3 Transformer Model

To optimise the transformer model, the hyperparameters were varied according to Table 2. The optimisation was carried out through 50 Optuna experiments using TPE sampling, having the same choice motivation as in Section 3.3.2.

To determine the search space in Table 2 we relied on previous studies as detailed further. The number of decoder layers was either 2 or 4, which is comparable to (Van Biljon et al., 2020; Lusito et al., 2023) which had between 2 and 12 encoder and/or decoder layers. Since Van Biljon et al. (2020) obtained better results for shallow and medium transformers while having larger datasets, we decided not to increase the number of decoder layers too much. The possible number of attention heads is in line with that of Lusito et al. (2023) who used 4 attention heads, however, we decided to include 2 attention heads in the search space for some variety, while keeping the model compact. Given the scarce dataset, the embedding dimension was kept to either 64 or 128, which is lower than what Lusito et al. (2023) used in their text normalisation study but comparable to the embedding dimensions proposed for the RNN models in the previous subsection, motivated by models performing well on a next-word prediction task. The optimiser, learning rate, batch size, and dropout rate were kept in the same range as the same Lusito et al. (2023) study, with some additions made to add some variety to the search space. Lastly, the patience was once again set semi-arbitrarily to 5, as in Section 3.3.2.

Hyperparameter	**Values**
Number of decoder layers	2, 4
Number of decoder attention heads	2, 4
Embedding dimension	64, 128
FFN embedding dimension	256, 512, 1024
Optimiser	Adam
Learning rate	10^{-3}, 10^{-4}, 10^{-5}
Batch size	16, 32
Dropout rate	0.2, 0.3, 0.4
Early stopping patience	5

Table 2: Hyperparameter values for transformer model optimisation.

3.4 Evaluation

The three optimal models were evaluated based on the three criteria below and the best-performing model was integrated into a mobile keyboard application.

1. **Top-3 Accuracy**: This type of accuracy measures the proportion of times when the expected next word is one of the top three predictions of the model. Since users will be presented with three possible predictions, this metric aligns with the user's expectations. The top-3 accuracy is defined in Equation 26, where N is the total number of test sequences, y_i is the expected i-th input, $\hat{Y}_3$ is the subset of the top three predicted words for the i-th input, and $\mathbf{1}_{\hat{Y}_3}(y_i)$ is the indicator function that outputs 1 if the expected next-word is in the predicted subset and 0 otherwise.
$$\text{Top-3 Accuracy} = \frac{\sum_{i=1}^{N} \mathbf{1}_{\hat{Y}_3}(y_i)}{N}, \text{ where } \hat{Y}_3 = \{\hat{y}_{i,1}, \hat{y}_{i,2}, \hat{y}_{i,3}\} \tag{26}$$

2. **Inference Time**: This criterion refers to the time that a model takes to generate a prediction given an input. Since the aim is to integrate the model into a mobile application, quick responses are essential to facilitate a positive user experience.

3. **Model Dimension**: For a model to be deployed on a mobile device, the model dimension should be kept minimal without sacrificing its accuracy. We want to ensure that the model is not only computationally efficient but also storage efficient.

3.5 Mobile Keyboard

Building upon the theoretical background presented in Section 2.6, we intended to develop a keyboard application which combines the needs of the users with the current industry standards. Thus, the conceptual design of the application followed two stages:

1. **Feature selection based on industry standards**: The keyboard proposed in this section is a QWERTY keyboard similar to the Google and Samsung keyboards (Google LLC, n.d.; Samsung Electronics Co., Ltd., n.d.) to ensure a rapid adjustment and predictability during use.

   The features available to the users are:

   (a) Access to digits and special characters through a dedicated button on the bottom-left side of the keyboard. The layout of the special characters follows the Samsung keyboard and can be consulted in Appendix A.

   (b) Text capitalisation, which is available by pressing the button above the special characters button. Users can capitalise one key by pressing the button once, or capitalise all future keys by pressing the button twice.

   (c) Next-word prediction, which follows both the Google and Samsung keyboards by displaying the three most probable words in a field above the main keys.

2. **Integrating the needs of the users**: After defining the core features and creating a draft of the layout, we consulted Dr Vijay D'Souza, the founding director of NEILAC (North Eastern Institute of Language and Culture, n.d.). In a meeting with Dr Vijay D'Souza, we were made aware of a previous project concerning a Hrusso Aka keyboard and its drawbacks. Firstly, the previous keyboard used long-press for accessing letters with diacritics, which users found was slowing them down and disrupting the typing flow as diacritics are common in Hrusso Aka. Secondly, the previous keyboard had an old-fashioned look which was not appealing to the youth.

   To overcome the first issue, our keyboard proposes a row of Hrusso Aka diacritics built into the main view, just above the first row of keys. The possible diacritics are the acute, the grave, the umlaut, the circumflex, and the tilde, with the possibility of combining the acute or grave with the umlaut. For a character with diacritics the user can either: *(i)* press the diacritic key(s) and then the letter key, *(ii)* press the letter key and then the diacritic key(s), or *(iii)* press one diacritic key, the letter key, and another diacritic key for letters which combine the acute or grave with the umlaut.

   Additionally, in the same meeting, Dr Vijay D'Souza specified that the vast majority of Hrusso Aka people are Android users, which was taken into account in the implementation stage of the keyboard.

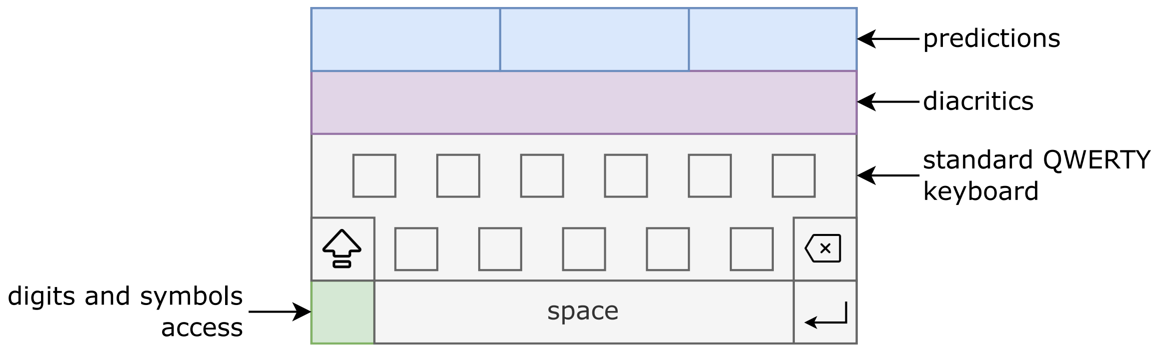


Figure 10: Keyboard design outline.

By adhering to the aforementioned industry standards and considering the user needs, the keyboard was implemented for Android devices and was structured around four main action areas. As illustrated in Figure 10, the keyboard has a next-word prediction area at the top, followed by a diacritics area, the main keys, and the button which changes the view towards the special characters in the bottom-left corner.

3.5.1 Implementation

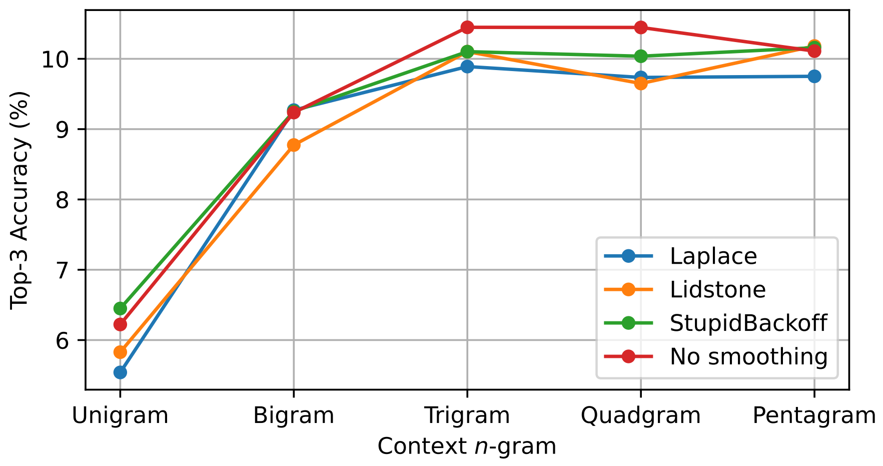
The keyboard was developed in Android Studio[2] with Kotlin as the programming language and Gradle as the build automation tool. The minimum software development kit (SDK) was set to version 24, which implies that the keyboard requires Android 7.0 (Nougat) or higher to run. While the latest SDK version is 34, and higher versions are preferable due to performance, memory, and battery usage improvements, they are not compatible with the vast majority of devices. Therefore, the use of SDK 24 ensures that the keyboard is compatible with approximately 97.4% of Android devices according to the Android Studio estimates.

[2]https://developer.android.com/studio

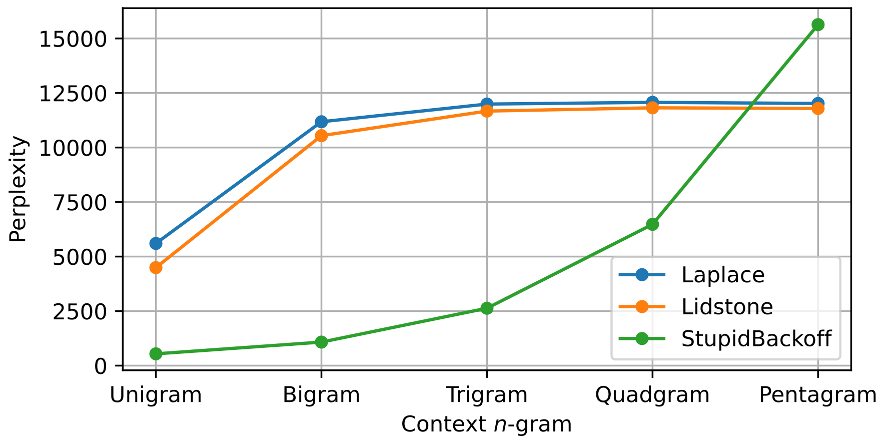
4 Results

This section presents and discusses the results of the experiments described in 3. First, the results of the *n*-gram hyperparameter optimision are presented, followed by the results of the RNN and transformer hyperparameter optimisation. Then the models are compared to determine the optimal architecture. Finally, an overview of the keyboard implementation is provided.

4.1 N-gram Hyperparameter Optimisation



(a) Top-3 validation accuracy as a function of time, differentiated based on the smoothing technique.



(b) Validation perplexity as a function of time, differentiated based on the smoothing technique.

Figure 11: Top-3 accuracy and perplexity plots of the *n*-gram models.

The results of the *n*-gram hyperparameter optimisation are depicted in Figure 11. In terms of the top-3 accuracy presented in Figure 11a, we can observe that the trigram models achieve the highest performance which is between 9.89% and 10.45%. For the quadgrams and pentagrams, we can observe a slight decrease in accuracy, which could be attributed to the increasing number of unseen *n*-grams as we increase the context window. Among the different types of models, the model without smoothing and the model with StupidBackoff smoothing maintained the highest performance no matter the order of the n-gram.

Regarding perplexity, depicted in Figure 11b, we can notice a general trend, namely, the perplexity increases with the order of the n-gram. Overall, the model using StupidBackoff achieved the lowest perplexity scores with the exception of the pentagram model. The highest perplexity was achieved by the model using no smoothing, which was not plotted because it was infinity due to encountering zero probabilities. As with the increase in accuracy for the quadgrams and pentagrams, the increase in perplexity is also likely a result of the increasing number of unseen n-grams, which increases the uncertainty of the model.

Overall, while higher-order n-gram models may improve the accuracy of a model, they also tend to increase its perplexity. Thus, we have selected the StupidBackoff trigram model as the optimal n-gram model due to the balance between higher accuracy and lower perplexity.

4.2 RNN Hyperparameter Optimisation

Figure 12 displays the Optuna optimisation history of the RNN model using the TPESampler. The validation loss, decreases most significantly in the first 10 trials and achieves a minimum validation loss of 6.35 after 41 trials.

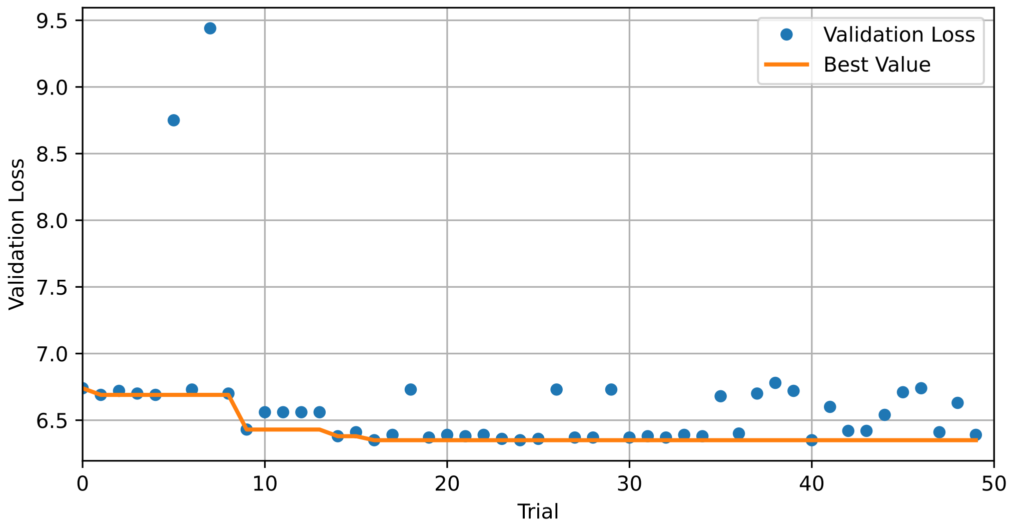


Figure 12: The TPESampler optimisation history of the RNN model.

To further understand how model performance is affected by each hyperparameter, Figure 13 illustrates the fANOVA importance of each hyperparameter during the optimisation of the RNN models. According to the fANOVA analysis, the most important hyperparameters are the optimiser, the learning rate and the unit type, responsible for 28%, 15% and 9% of the validation loss variation, respectively. The other hyperparameters, i.e. the batch size, embedding dimension and number of units in the hidden layer are less important, each being responsible for less than 6% of the validation loss variation.

The optimal hyperparameters for the RNN resulting from the best trial can be observed in Table 3. The best performing model is a GRU-based model with one layer of 512 units optimised using Adam, a learning rate of 10^{-3}, a batch size of 16, an embedding dimension of 50, and a patience of 5 epochs.

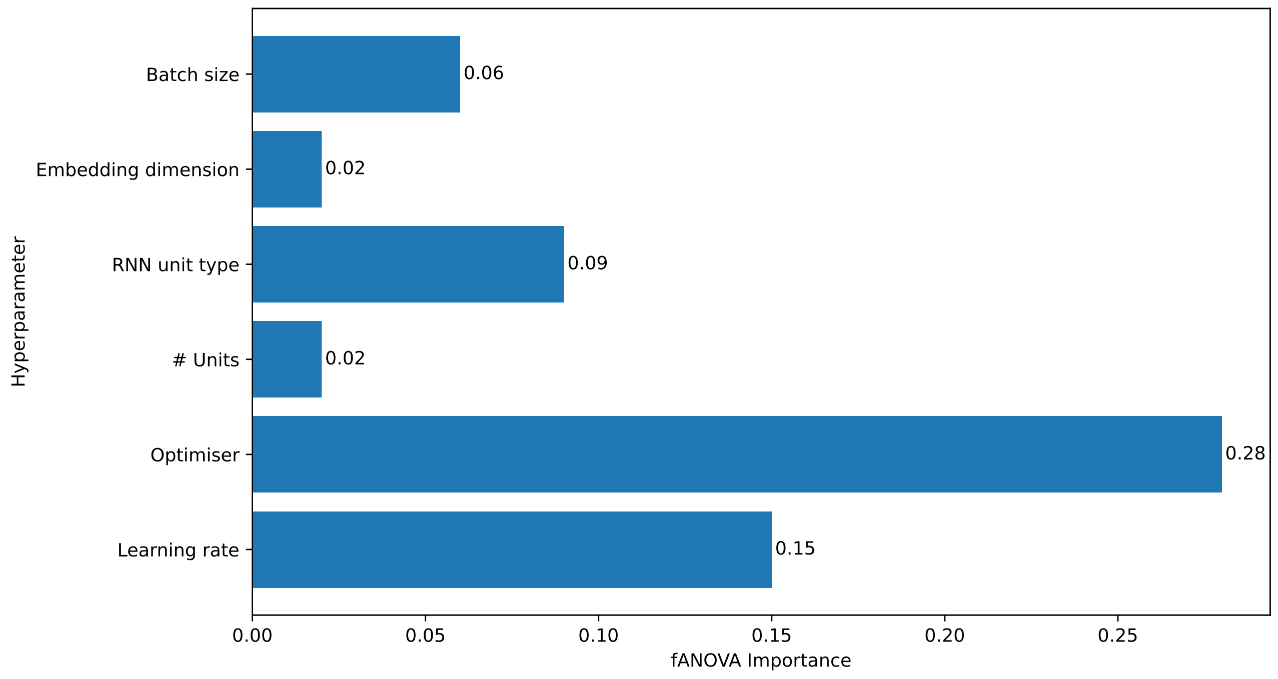


Figure 13: Hyperparameter importance in RNN optimisation.

The training and validation loss curves are presented in Figure 17 to check for any possible overfitting or underfitting. Both of the losses decline for the first three epochs. Afterwards the training loss continues steadily decreasing, while the validation loss increases. The optimal model was saved after the third epoch when there are no signs of overfitting with the training and validation loss being almost equal. Normally, we might argue that there is underfitting due to the high loss, but in our case, the more complex LSTM model did not perform better. An additional argument against underfitting is that Hard et al. (2018) successfully used only one hidden layer and a comparable number of hyperparameters for a much larger dataset which inherently contains more complex relationships.

Hyperparameter	**Values**
Embedding dimension	**50**, 100
RNN type	LSTM, **GRU**
Number of hidden layers	**1**
Number of units in hidden layer	128, 256, **512**
Optimiser	SGD, **Adam**
Learning rate	**1e-3**, 1e-4
Batch size	**16**, 32, 64
Early stopping patience	**5**

Table 3: Hyperparameter values for RNN model optimisation, where the optimal hyperparam- eters are marked in bold.

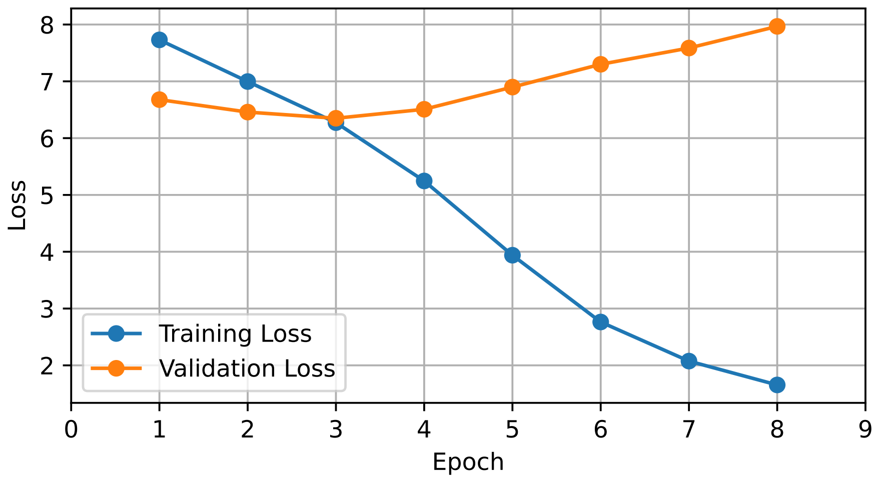


Figure 14: Training and validation losses of the optimal RNN model over the epochs.

4.3 Transformer Hyperparameter Optimisation

Figure 15 displays the Optuna optimisation history of the model using the TPESampler. The objective value (i.e. validation loss) significantly decreases in the initial trials. The study then converges and achieves the best performance for trial 34, where the validation loss is 10.074.

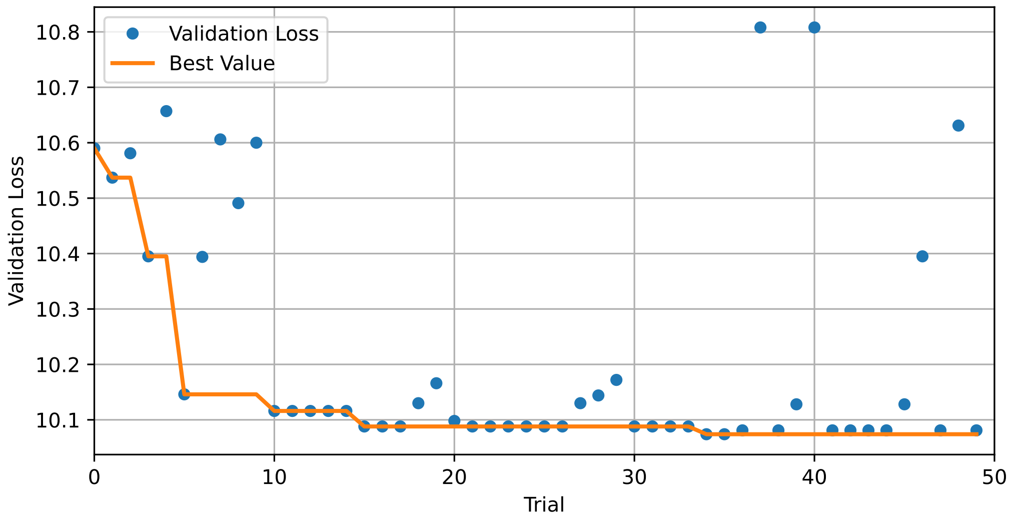


Figure 15: The TPESampler optimisation history of the transformer model.

Figure 16 illustrates the fANOVA importance of each hyperparameter towards achieving the optimal model. According to this assessment, the most important hyperparameters are the learning rate and the embedding dimension of the decoder, where they are responsible for 30% and 17% of the validation loss variation, respectively. Other hyperparameters such as the batch size, number of attention heads, the FFN embedding dimensin, and the number or decoder layers are only marginally important, each being responsible for less than 4% of the validation loss variation.

The optimal hyperparameters resulting from the best trial can be observed in Table 4. The best performing model is a compact model which consists of 2 decoder layers, 2 attention head, an embedding

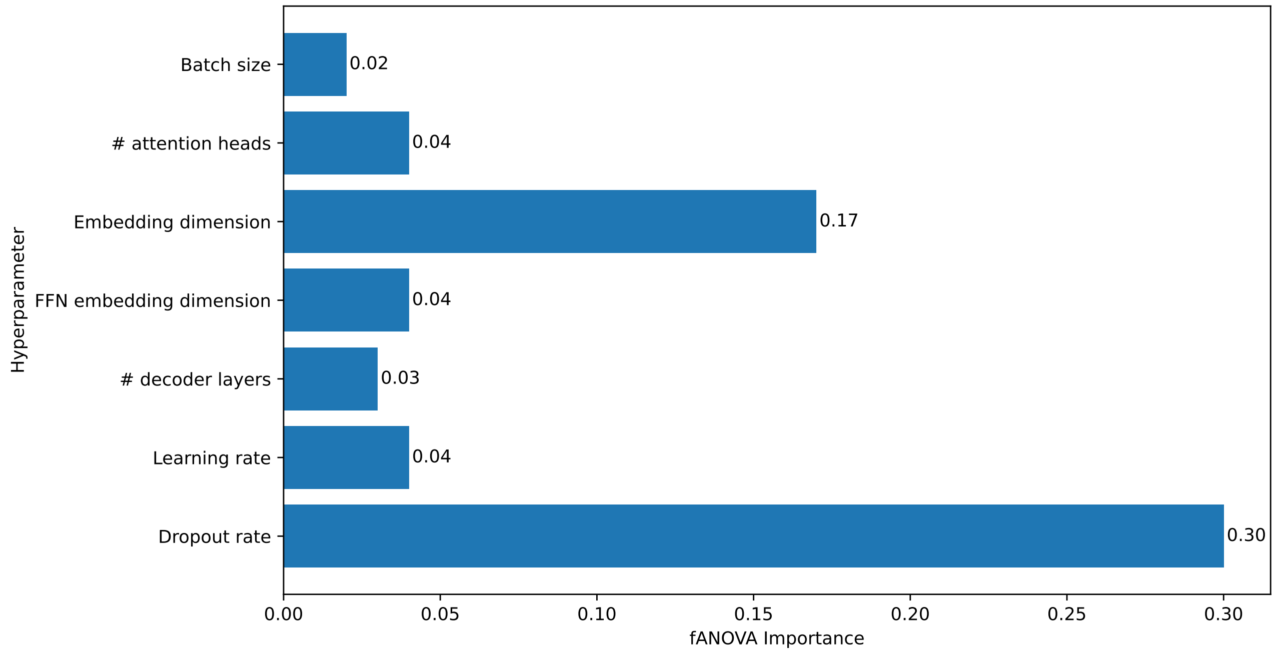


Figure 16: Hyperparameter importance in transformer optimisation.

dimension of 64, a FFN embedding dimension of 1025. Other optimal values include a learning rate of 10^{-5}, a batch size of 16, and a dropout rate of 0.2.

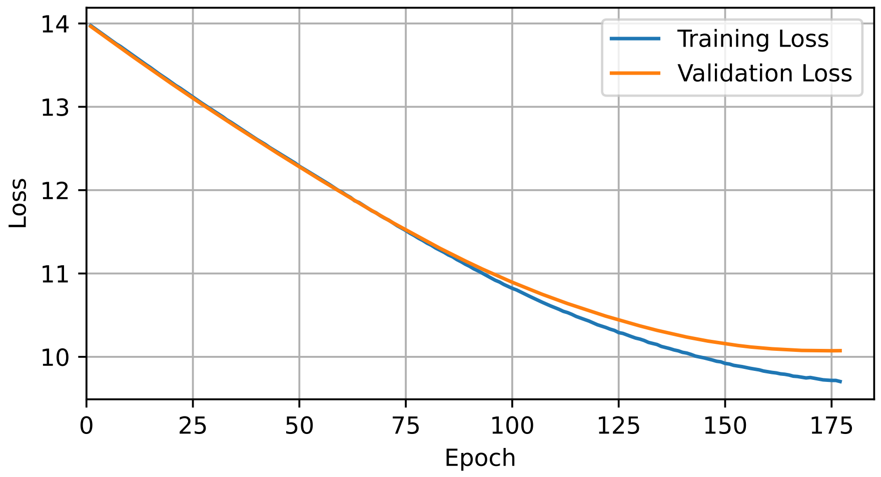


Figure 17: Training and validation losses of the transformer model over the epochs.

To assess the possibility of overfitting and underfitting, the training and validation loss curves are presented in Figure 17. Both of the losses decrease steadily and almost linearly until epoch 75. In the last 100 epochs, the training loss continues to decrease at a faster rate than the validation loss, which converges at epoch 171. Since the training loss is not much lower, there is no strong indication of possible overfitting. Additionally, both losses are converging at around 10, which is typically quite high for cross-entropy loss. However, since more complex models did not lead to lower losses, there is no clear evidence of underfitting.

Hyperparameter	**Values**
Number of decoder layers	**2**, 4
Number of decoder attention heads	**2**, 4
Embedding dimension	**64**, 128
FFN embedding dimension	256, 512, **1024**
Optimiser	**Adam**
Learning rate	1e-3, 1e-4, **1e-5**
Batch size	**16**, 32
Dropout rate	**0.2**, 0.3, 0.4
Early stopping patience	**5**

Table 4: Hyperparameter values for transformer model optimisation, where the optimal hyperparameters are marked in bold.

4.4 Optimal Model Comparison

The performance comparison on the test dataset of the three optimised models previously presented is summarised in Table 5. The RNN GRU-based model attained the highest top-3 accuracy at 19.94%, followed by the trigram model at 9.74%, and the transformer model at 7.03%. The GRU model was also notably much faster than the other two models, making it the overall preferred model both in terms of accuracy and inference time.

Method	**Top-3 Accuracy**	**Inference Time**	**Model Dimension**
Trigram with StupidBackoff	9.64%	0.01 s	**Negligible**
GRU	**19.94%**	**0.0006 s**	8 521 978 params.
Transformer	7.03%	0.04 s	7 449 728 params.

Table 5: Performance comparison of optimised models.

To ensure accurate inference time estimates, the evaluation of each method was conducted on the same device, specifically a machine equipped with one NVDIDIA A40 GPU, 9 vCPUs, and 50 GB of RAM.

4.5 Keyboard Implementation

To support on-device inference, the optimal GRU-based model, which was implemented in TensorFlow, was converted using TensorFlow Lite[3]. The resulting model measures only 8.91 MB and has an on-device inference time of approximately 0.009s.

The keyboard was successfully implemented according to the methodology specified in Section 3.5. A visual representation of the final main view can be consulted in Figure 18.

[3]`https://www.tensorflow.org/lite`

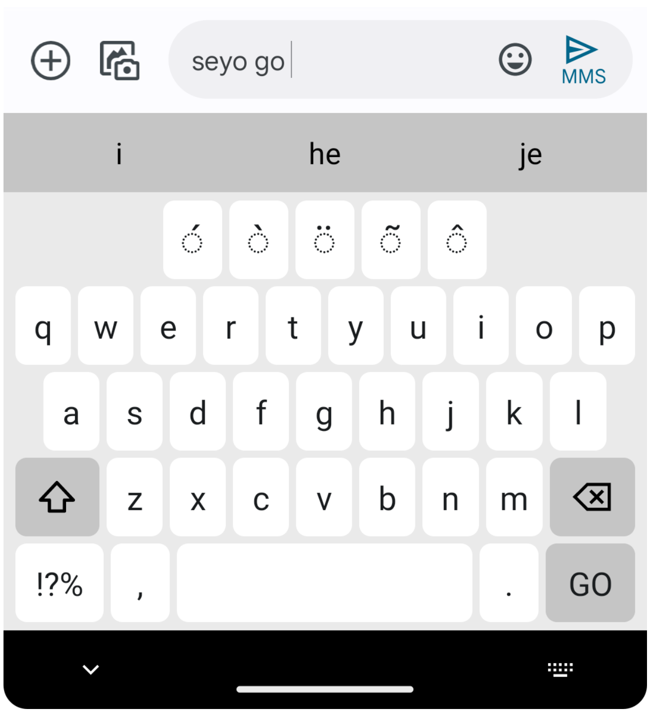


Figure 18: Screenshot of the implemented keyboard.

5 Conclusions

This thesis focused on exploring the next-word prediction model in the context of the Hrusso Aka language from Northeast India. Hrusso Aka is one of the many endangered languages (Sutherland, 2003; UNESCO, 2024), which could benefit from language preservation and revitalisation programmes aimed at encouraging native speakers to communicate in their mother tongue. This project aimed to promote the usage of Hrusso Aka in the daily life of the Hrusso Aka people through the development of a next-word prediction model integrated into a mobile keyboard.

Although language models have become ubiquitous in our daily lives, from ChatGPT (OpenAI, 2022) and Amazon's Alexa to our daily smartphone keyboards which output likely words or emojis, these technologies are often not available for low-resource languages. Most of the advancements in NLP are concentrated around only 20 languages (Magueresse et al., 2020), which causes further technological alienation for the already few people speaking endangered languages.

Developing any language models for (very) low-resource languages comes with the challenge of sparse data, often with a limited scope which may or may not match the task at hand. In the case of Hrusso Aka, the most extensive dataset today consists of less than 100 000 tokens, almost infinitesimally less compared to the 7.5 billion token dataset used by Google to train their next-word prediction model for the English language (Hard et al., 2018). Additionally, the Hrusso Aka language is not closely related to other languages, being an isolate does not make it a good candidate for transfer learning. As a further challenge, the Hrusso Aka dataset is not an accurate representation of mobile written text. The dataset comprises transcripts of spoken communication mostly around the topics of family life, the local environment, or traditional occupations, as presented earlier in Figure 8.

In an attempt to overcome these significant challenges, alongside the limited literature on next-word prediction models for (very) low-resource languages, we explored a variety of models ranging widely in architecture and complexity. Specifically, we experimented with numerous configurations of *n*-gram, RNN, and transformer models. As a consequence, this work does not only aim to identify the best-performing model for Hrusso Aka but also offers insights and a starting point to others facing similar challenges in (very) low-resource language settings.

With these ideas in mind, we set two goals at the beginning of this thesis, which we can now address as follows:

1. *Answering the primary research question:*

   What are the comparative performances of statistical models, RNN models and transformers in Hrusso Aka next-word prediction?

   Throughout this thesis, we have discussed and observed the task of next-word prediction in the Hrusso Aka language through three types of models of increasing complexity, namely, *n*-gram, RNN and transformer models. Our analysis has shown that all three models can learn to perform next-word prediction to an extent. RNN models, particularly GRU-based models, outperformed both the *n*-gram and transformer models in terms of accuracy and inference time, obtaining a top-3 accuracy of 19.94% and an inference time of 0.0006 s. In contrast, the *n*-gram and transformer models achieved an accuracy of less than 10% each, while being several orders of magnitude slower.

A top-3 accuracy of 19.94% achieved using the RNN GRU-based model is rather promising, especially when we take into account that Hard et al. (2018) obtained a top-3 accuracy of around 27% also using a compact RNN model but while capitalising on a 7.5 billion tokens compared to less than 100 000 tokens in our case. Of course, it is worth noting that the Hrusso Aka dataset is out-of-scope for a written mobile task, while the dataset utilised by Hard et al. (2018) had mostly originated from chat applications. Nonetheless, this thesis proves that GRU-based models are capable of learning in very low-resource scenarios, constituting a good choice for (very) low-resource language settings.

One pattern we have consistently observed is that the performance in each model type decreased with model complexity. In particular, the top performer of the *n*-gram models was the trigram with StupidBackoff smoothing, which can adapt the probability estimates of unseen context by reducing the number of words in the context until finding a known probability. For RNN models, the GRU performed better than the more complex LSTM. Whereas for the transformer model, the more compact model with only two decoder layers and two decoder attention heads outperformed more complex configurations. Since this trend is in line with previous studies such as the ones performed by Van Biljon et al. (2020) and Araabi and Monz (2020) in neural machine translation tasks, we have reasonable grounds to suggest that we should aim for less complex transformer models when dealing with limited datasets.

and the secondary research question:

What are the scalability differences between statistical models, RNNs, and transformer models in real-world use?

2. *Deploying the best-performing next-word prediction model into a mobile keyboard application.* The best-performing model was then successfully converted to an on-device model of only 8.91 MB using TensorFlow Lite. The next-word prediction model was integrated into a mobile keyboard for Android devices, which are the most common in the Hrusso Aka community according to Dr Vijay D'Souza, our point of contact and the founding director of NEILAC (North Eastern Institute of Language and Culture, n.d.). We also ensured that the drawback of the first iteration of a Hrusso Aka keyboard, which was the disruption of typing flow by accessing diacritics using long-press was eliminated. Instead, in our application proposal, users have access to dedicated buttons for typing out diacritics. One limitation of the present work is that the application was not tested or developed in collaboration with the users, which was detailed in Section 5.1.2.

   As a result, we obtained a responsive, modern, and user-friendly keyboard which follows the Android design industry standard set by Google LLC (n.d.) and (Samsung Electronics Co., Ltd., n.d.). Embarking and completing this end-to-end project, while working with a scarce dataset, serves as proof that dedicating our resources towards developing tools for endangered languages can meaningfully contribute towards the preservation and revitalisation of such languages.

   In the pursuit of technological advancements and pushing the limits of state-of-the-art approaches in NLP, we should not neglect low-resource languages, as that further accentuates their decline and the knowledge gap with their high-resource counterparts.

5.1 Limitations and Future Work

5.1.1 Next-Word Prediction Model

At this moment, the accuracy and reliability of our models cannot be fully benchmarked due to the limitations of the training data, which was scarce and most likely out-of-domain. While the dataset covered a variety of topics, it primarily focused on traditional occupations and the surrounding environment. As a result, it remains unclear to what extent users would communicate about these topics through text. Furthermore, differences in demographic factors, such as the age of the users, might lead to additional divergence in topic choice, further hindering the generalisability of our results in real-life messaging scenarios.

We propose two potential improvement plans depending on the resources available to either improve the system proposed in this thesis or to develop and improve similar systems for other endangered languages:

1. **Federated learning.** One of the most efficient ways of gathering more data and updating the model accordingly is implementing federated learning. This would allow many users to generate data while using the keyboard, which would significantly improve the performance of the model. The drawback is that this approach could be costly since one would need to fund the maintenance of the overall system and the server associated with the model updates. For small communities and organisations which operate with limited funds, the costs could be a significant barrier.

2. **Two-stage rollout.**

   - **Stage 1: Volunteer Data Collection.** Initially the application can be rolled out as is to volunteers who are willing to generate data for further training.

     This approach would not require substantial funds, as the GRU model provided in this thesis which provides the most hoped could be retrained or fine-tuned using the newly generated data.

   - **Stage 2: Final Model Rollout.** Once the model has been fine-tuned on the data collected from the volunteers, it can be released in the final iteration of the application. Unlike the federated learning approach, which allows for continuous model improvement, this approach would be static or would require other data collection efforts.

   In the final stage, it is still possible to perform periodical updates locally using non-sensitive data generated by the users. In this manner, the quality of the predictions can increase by adapting to the choice of words of individual users.

5.1.2 Mobile Keyboard

One limitation of developing the mobile keyboard is that the users were involved only at the beginning of the development stage. The main concern is that the development process did not feature a testing stage where users could provide their feedback and satisfaction with the final application. Overall, one should not expect significant dissatisfaction with the main features of the keyboard such as the layout, the position of the prediction bar or access to the special characters, as they follow the same standards as other applications which are widely used. However, our proposal diverges from popular approaches when it comes to handling diacritics. Unlike popular keyboards where access to characters with diacritics is achieved through a long-press on the desired main letter, our keyboard features access through a combination of keystrokes between the diacritic and the main letter. This change was found in the initial user feedback provided by North Eastern Institute of Language and Culture (n.d.) but user satisfaction surveys should be conducted to assess whether our approach was successful. At this moment, it is unclear whether potential users would value the increased typing flow of separate diacritic buttons over the habitual environment that long press diacritics offer. To settle this issue, one could roll out two versions of the keyboard for testing and assess user satisfaction.

The keyboard could be further improved by incorporating a variety of additional features. In our opinion, the initial improvements should focus on enhancing the accessibility of the application and should strive to appeal to younger generations, who are most at risk of not speaking Hrusso Aka. Some potential examples include:

- **Emoji support.** Currently, the keyboard does not offer the possibility of using emojis. While emojis are not essential for tasks such as writing an email or browsing the internet, they have become part of our daily lives in messaging applications and on social media. Emojis are especially important for the younger generations, and their absence might discourage the youth from using the Hrusso Aka keyboard in favour of a different keyboard where emojis can enhance their communication.
- **Dark mode support.** In addition to accommodating user-specific aesthetic preferences and aligning with other applications which offer both light and dark modes, providing a dark mode would greatly improve the accessibility and user experience. Firstly, darker colours can improve user experience by improving readability in low-light conditions and reducing eye strain. Secondly, dark mode can provide increased support to individuals suffering from photophobia. Lastly, an additional benefit is that it could lead to battery savings, as displaying dark pixels uses less energy than displaying light pixels.
- **Speech-to-text.** This feature is not as simple to implement as the other two and would require the development of a dedicated model. Nonetheless, its development and integration would improve the accessibility of the application for users who may have difficulties typing on their phones. An additional benefit is that it could encourage native Hrusso Aka speakers with good speaking proficiency but limited writing experience to start using the application, increasing their engagement with the language, and hopefully improving their confidence to write over time.

5.2 Final Recommendations

To conclude this project, we would like to offer a set of recommendations to those wanting to develop a next-word prediction model as a means of language revitalisation.

1. **Get to know the situation of the language within the speaker community.**
   It is important to set a target demographic and a target task for any language model. Even more for endangered languages, whose datasets are often quite limited and not complex enough for

a high-performing general language model. Thus, to ensure relevant predictions, it would be preferred if the collected data in the dataset and the use case fall under the same scope.

2. **Get to know your dataset.**
   The more you know about your data, the better you can set achievable goals. Consider the complexity of your dataset, as often less complex models perform better when dealing with scarce data. Additionally, if your language of choice is not an isolate, you might want to consider transfer learning by leveraging other higher-resource languages from the same language family.
3. **Experiment within the limits of your resources and do not aim for perfection.**
   If you wish to develop a next-word prediction model for an endangered language, it is unlikely to come across much previous research. Thus, alongside some informed decisions which can be drawn from literature, one must also rely on trial and error.

Finally, we have a suggestion even for those whose interests in NLP are not solely set on (very) low-resource languages. The next time you experiment with a new language model comparing its performance on multiple languages, consider also evaluating it on one of the many endangered language datasets on OPUS [4]. While the final performance might not be as impressive as choosing a dataset for a high-resource language, your contribution could positively influence the survival of one of the many languages that are currently at risk.

[4]`https://opus.nlpl.eu/`

Bibliography

7000 Languages. (2024). *7000 languages.* Retrieved from `https://www.7000.org/` (Accessed: 2024-08-03)

Abras, C., Maloney-Krichmar, D., Preece, J., et al. (2004). User-centered design. *Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications*, *37*(4), 445–456.

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). *Optuna: A next-generation hyperparameter optimization framework.* Retrieved from `https://arxiv.org/abs/1907.10902`

Araabi, A., & Monz, C. (2020). Optimizing transformer for low-resource neural machine translation. *arXiv preprint arXiv:2011.02266*.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Balcan, M. F., Blum, A., Fine, S., & Mansour, Y. (2012). Distributed learning, communication complexity and privacy. In *Conference on learning theory* (pp. 26–1).

Balouch, S. (2024). Prediction of Next Word in Balochi Language Using N-gram Model. *Sukkur IBA Journal of Computing and Mathematical Sciences*.

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, *24*.

Blench, R., & Post, M. (2011). *(de)Classifying Arunachal languages: Reconsidering the evidence.* (Unpublished manuscript)

Blench, R., & Post, M. W. (2014). Rethinking sino-tibetan phylogeny from the perspective of north east indian languages. *Trans-Himalayan Linguistics*, *266*, 71–104.

Bodt, T. A., & Lieberherr, I. (2015). First notes on the phonology and classification of the bangru language of india. *Linguistics of the Tibeto-Burman Area*, *38*(1), 66–123.

Brants, T., Popat, A., Xu, P., Och, F. J., & Dean, J. (2007). Large language models in machine translation. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (emnlp-conll)* (pp. 858–867).

Bromham, L., Dinnage, R., Skirgård, H., Ritchie, A., Cardillo, M., Meakins, F., … Hua, X. (2022). Global predictors of language endangerment and the future of linguistic diversity. *Nature ecology & evolution*, *6*(2), 163–173.

Cauchy, A.-L. (1847). Méthode générale pour la résolution de systèmes d'équations simultanées. *Compte rendu de séances de l'académie des sciences*, *25*, 536-538.

Chen, X., Xie, H., & Tao, X. (2022). *Vision, status, and research topics of natural language processing* (Vol. 1). Elsevier.

Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Cooper, A. (1995). *About Face: The Essentials of User Interface Design*. Wiley.

Crystal, D. (2000). *Language Death*. Cambridge University Press.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182-197. doi: 10.1109/4235.996017

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

D'Souza, V. A. (2015). *Documentation and Description of the Hrusso Aka Language of Arunachal Pradesh*. Endangered Languages Archive. (Handle: `http://hdl.handle.net/2196/00-0000-0000-000F-BF58-9`. Accessed on [November 25, 2024])

D'Souza, V. A. (2021a). *Aspects of Hrusso Aka Phonology and Morphology* (Unpublished doctoral dissertation). University of Oxford.

D'Souza, V. A. (2021b). *Aspects of hrusso aka phonology and morphology* (Doctoral dissertation). University of Oxford.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, *12*(7).

Eberhard, D. M., Simons, G. F., & Fennig, C. D. (2019). *Ethnologue: Languages of the world* (22nd ed.). SIL International. Retrieved from `https://www.ethnologue.com/`

Elan [computer software]. (n.d.). Nijmegen: Max Planck Institute for Psycholinguistics, The Language Archive. Retrieved from `https://archive.mpi.nl/tla/elan` (Retrieved from `https://archive.mpi.nl/tla/elan`)

Endangered Languages Documentation Programme. (2024). *Endangered languages documentation programme*. Retrieved from `https://www.eldp.net/` (Accessed: 2024-08-03)

Gale, W. A., & Church, K. W. (1994). What's wrong with adding one. *Corpus-based research into language: In honour of Jan Aarts*, 189–200.

Gao, D., Ju, C., Wei, X., Liu, Y., Chen, T., & Yang, Q. (2019). Hhhfl: Hierarchical heterogeneous horizontal federated learning for electroencephalography. *arXiv preprint arXiv:1909.05784*.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. (`http://www.deeplearningbook.org`)

Google LLC. (n.d.). *Gboard – the google keyboard (version 14.4.07.646482735-release-arm64-v8a)*. Retrieved from `https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin&hl=en` (Accessed: 2024-08-17)

Hamarashid, H. K., Saeed, S. A., & Rashid, T. A. (2021). Next word prediction based on the n-gram model for kurdish sorani and kurmanji. *Neural Computing and Applications*, *33*, 4547–4566.

Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., ... Ramage, D. (2018). Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hutter, F., Hoos, H., & Leyton-Brown, K. (2014, June). An efficient approach for assessing hyper-parameter importance. In *Proceedings of international conference on machine learning 2014 (icml 2014)* (p. 754–762).

Jelinek, F. (1998). *Statistical methods for speech recognition.* MIT press.

Jelinek, F., & Mercer, R. L. (1980). Interpolated estimation of markov source parameters from sparse data. In E. S. Gelsema & L. N. Kanal (Eds.), *Proceedings, workshop on pattern recognition in practice* (pp. 381–397). North Holland.

Jurafsky, D., & Martin, J. H. (2019). *Speech and Language Processing* (3rd ed.). Stanford University.

Katinskaia, A., Nouri, J., & Yangarber, R. (2017). Revita: a system for language learning and supporting endangered languages. In *Proceedings of the joint workshop on nlp for computer assisted language learning and nlp for language acquisition* (pp. 27–35).

Katz, S. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE transactions on acoustics, speech, and signal processing*, *35*(3), 400–401.

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization. In *Proceedings of the international conference on learning representations (iclr).* Retrieved from `https://arxiv.org/abs/1412.6980`

Konečný, J., McMahan, B., & Ramage, D. (2015). Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*.

Konečnỳ, J., McMahan, H. B., Ramage, D., & Richtárik, P. (2016). Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*.

Konow, S. (1902). Note on the languages spoken between the assam valley and tibet. *Journal of the Royal Asiatic Society of Great Britain and Ireland*, 127–137. Retrieved 2024-07-24, from `http://www.jstor.org/stable/25208376`

Lankford, S., Afli, H., & Way, A. (2024). Transformers for low-resource languages: Is f\'eidir linn! *arXiv preprint arXiv:2403.01985*.

Lusito, S., Ferrante, E., & Maillard, J. (2023). Text normalization for low-resource languages: The case of ligurian. In *Proceedings of the sixth workshop on the use of computational methods in the study of endangered languages* (pp. 98–103).

Magueresse, A., Carles, V., & Heetderks, E. (2020). Low-resource languages: A review of past work and future challenges. *arXiv preprint arXiv:2006.07264*.

Moseley, C. (2010). *Atlas of the world's languages in danger*. UNESCO.

Mozer, M. (1995, 01). A focused backpropagation algorithm for temporal pattern recognition. *Complex Systems*, *3*.

Norman, D. A. (1986, 01). Cognitive Engineering. In (p. 31-61). Lawrence Erlbaum Association. doi: 10.1201/b15703-3

Norman, D. A. (1988). *The Psychology of Everyday Things*. Basic Books.

North Eastern Institute of Language and Culture. (n.d.). *Neilac – north eastern institute of language and culture*. `https://www.neilac.org.in/`. (Accessed: 2024-07-17)

OpenAI. (2022). *Chatgpt*. Retrieved from `https://openai.com/chatgpt` (Accessed: 2024-07-29)

Preece, J., Rogers, Y., & Sharp, H. (2002). *Interaction Design: Beyond Human-Computer Interaction*. New York, NY: John Wiley & Sons.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training.

Rakib, O. F., Akter, S., Khan, M. A., Das, A. K., & Habibullah, K. M. (2019). Bangla word prediction and sentence completion using gru: an extended version of rnn on n-gram language model. In *2019 international conference on sustainable technologies for industry 4.0 (sti)* (pp. 1–6).

Robbins, H., & Monro, S. (1951). A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, *22*, 400–407.

Robinson, A. J., & Fallside, F. (1987). *The utility driven dynamic error propagation network* (Tech. Rep. No. CUED/F-INFENG/TR.1). Cambridge, UK: Engineering Department, Cambridge University.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533–536.

Samsung Electronics Co., Ltd. (n.d.). *Samsung keyboard (version 3.5.04.2 (2019.09.10))*. Retrieved from `https://galaxystore.samsung.com/prepost/000004406763?appId=com.sec.android.inputmethod` (Accessed: 2024-08-17)

Shafer, R. (1947). Hruso. *Bulletin of the School of Oriental and African Studies*, *12*(1), 184–196. doi: 10.1017/S0041977X00079994

Shamir, O., Srebro, N., & Zhang, T. (2014). Communication-efficient distributed optimization using an approximate newton-type method. In *International conference on machine learning* (pp. 1000–1008).

Shneiderman, B. (1987). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison Wesley.

Simons, G. F. (2019). Two centuries of spreading language loss. *Proceedings of the Linguistic Society of America*, *4*, 27–1.

Sinha, N., & Barbora, M. (2021). Language endangerment amongst Hruso-Aka and Koro. *Vaak Manthan*, *6*(II), 1–16.

Sutherland, W. J. (2003). Parallel extinction risk and global distribution of languages and species. *Nature*, *423*(6937), 276–279.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*.

Tieleman, T., & Hinton, G. (2012). *Lecture 6e-rmsprop: Divide the gradient by a running average of its recent magnitude.* pp. 26-31. (COURSERA: Neural Networks for Machine Learning)

Trabant, J. (2000). How relativistic are Humboldt's "Weltansichten"? In M. Pütz & M. Verspoor (Eds.), *Explorations in Linguistic Relativity* (pp. 25–44). Amsterdam: John Benjamins Publishing Company.

UNESCO. (2024). *Languages — UNESCO World Atlas of Languages.* Retrieved from `https://en.wal.unesco.org/discover/languages` (Accessed: 2024-07-21)

Van Biljon, E., Pretorius, A., & Kreutzer, J. (2020). On optimal transformer depth for low-resource language translation. *arXiv preprint arXiv:2004.04418*.

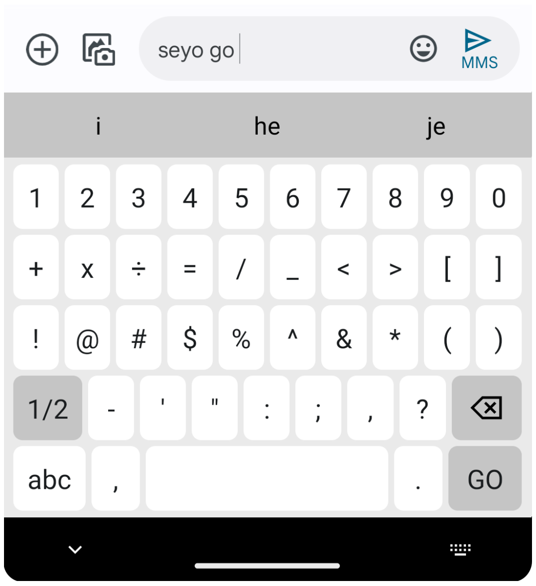
van Driem, G. (2007, 05). Endangered languages of South Asia. In (p. 303-341). doi: 10.1515/9783110905694-016

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

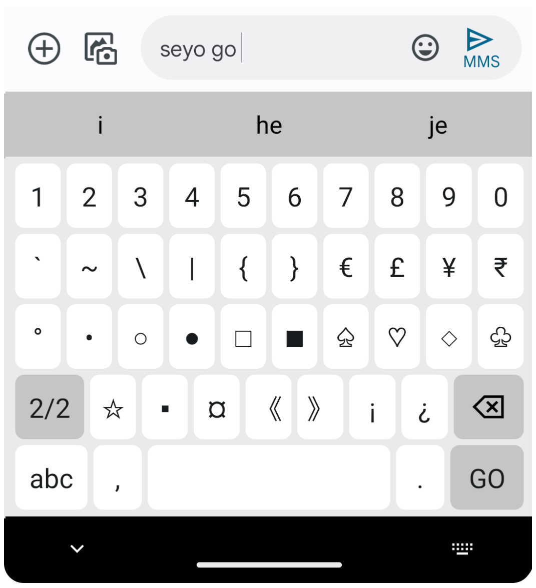
Zhang, S., Frey, B., & Bansal, M. (2022). How can nlp help revitalize endangered languages? a case study and roadmap for the cherokee language. *arXiv preprint arXiv:2204.11909*.

Appendices

A Special Characters



(a) The first page of the keyboard view for special characters.



(b) The second page of the keyboard view for special characters.

Figure 19: Screenshots of the implemented view for special characters following the Samsung keyboard layout (Samsung Electronics Co., Ltd., n.d.).