Bachelor's Thesis

Bringing Neural Spelling Correction to Mobile Keyboards Using a Client-Server Architecture

Hagen Tilmann Mogalle

August 23, 2025

Submitted to the University of Freiburg
IIF – Department of Computer Science
Chair for Algorithms and Data Structures

universität freiburg

University of Freiburg
IIF – Department of Computer Science
Chair for Algorithms and Data Structures

Author Hagen Tilmann Mogalle,
Matriculation Number: 4346709

Editing Time June 03, 2025 - September 03, 2025

Examiners Prof. Dr. Hannah Bast,
IIF – Department of Computer Science
Chair for Algorithms and Data Structures

Supervisor M.Sc. Sebastian Walter,
IIF – Department of Computer Science
Chair for Algorithms and Data Structures

Declaration I hereby declare, that I am the sole author and composer of this Thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.

I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Freiburg, 23.09.2025
Place, Date

HMogalle
Signature

Abstract

Mobile keyboards rely heavily on intelligent spelling correction to support users during text input. Modern neural networks deliver high accuracy but often exceed the computational limits of mobile devices. This thesis investigates the practical feasibility of employing a neural spelling correction model for mobile keyboards using a client-server architecture. We developed an Android keyboard application that retrieves corrections from a remote server and evaluated it in technical experiments and a user study. The technical evaluation shows that the system achieves an average end-to-end latency of 128 ms and a correction accuracy that clearly outperforms Google's Gboard. In the user study, we evaluated the acceptance of the system for both word-level and sentence-level correction. Both modes were found to be feasible in practice, with sentence-level correction being rated particularly positive.

Zusammenfassung

Mobile Tastaturen setzen in hohem Maße auf intelligente Rechtschreibkorrektur, um Nutzer bei der Texteingabe zu unterstützen. Moderne neuronale Modelle bieten eine hohe Korrekturquote, überschreiten jedoch häufig die Rechenleistung mobiler Geräte. Diese Arbeit untersucht die praktische Umsetzbarkeit von neuronaler Rechtschreibkorrektur für mobile Tastaturen durch eine Client-Server-Architektur. Hierzu wurde eine Android-Tastatur entwickelt, die Korrekturen von einem Server abruft. In technischen Experimenten und einer Nutzerstudie evaluieren wir das System. Die technische Evaluation zeigt, dass das System eine durchschnittliche End-to-End-Latenz von 128 ms erreicht und die Korrekturqualität deutlich über der von Googles Gboard liegt. In der Nutzerstudie haben wir die Akzeptanz des Systems für Wortkorrekturen und Satzkorrekturen bewertet. Beide Modi erwiesen sich in der Praxis als praktikabel, wobei Satzkorrekturen besonders positiv bewertet wurden.

Contents

Chapter 1

Introduction

Whether for chatting with friends or writing emails, smartphones have become a central tool of communication in our everyday life. Yet, this mobility brings new challenges. Virtual keyboards are far smaller than physical keyboards which makes them prone for spelling errors. Users often hide the characters with their fingers making it difficult to see, where exactly they are tapping. The „fat-finger problem“, where unprecise touches cause errors, illustrates the issue of small character keys. External factors such as walking while typing or a reduced thumb reach when holding the phone with only one hand can further increase error rates.

Spelling correction systems are therefore essential in mobile keyboards and users have high expectations for their accuracy. Traditional methods were often based on dictionaries or statistical n-gram models but reach their limits with complex errors. Neural models in contrast, have delivered impressive results and current state-of-the-art systems rely on them.

Yet neural models come with trade-offs. Running large models directly on mobile devices is often infeasible due to memory, battery and compute limitations[1]. As a result, most keyboards with local correction stick to lightweight models and ignore the potential of more powerful neural networks. Client-server architectures have

emerged as a practical solution. They outsource heavy computation to powerful servers. The keyboard sends text to a server, where the text is corrected and sent back. This allows for larger correction models, while keeping the computation on the keyboard minimal. Outsourcing computationally expensive tasks is not a new concept. We have seen generative AI systems, such as Dall-E and Claude, or voice assistants, such as Siri or Alexa already employ this architecture successfully. At the same time, network communication inherently introduces latency.

This thesis investigates whether a client-server architecture is feasible for mobile spelling correction while maintaining acceptable usability. We developed an Android keyboard, measured the end-to-end latency of the system and compared the correction quality to Google's Gboard keyboard. We conducted an empirical user study, to assess user acceptance in two correction modes. Acceptance of at least one mode would provide evidence that a client-server architecture is feasible for mobile spelling correction.

Chapter 2

Related Work

This section presents prior work in spelling correction and its application in mobile keyboards. We first summarize general approaches and follow with research on on-device and server-based mobile keyboards.

2.1 Spelling Correction in NLP

Spelling Correction in Natural Language Processing Spelling correction is a classical task in Natural Language Processing (NLP) and has been addressed using various methods. Early approaches were dictionary-based and used edit-distance algorithms to find similar valid words [2], [3]. These methods work well for non-word errors (e.g. eror → error), but fail to detect real-word errors (e.g. live → life) due to a lack of context information.

Statistical models, especially n-grams, addressed this gap by introducing limited word context [4]. They estimate the likelihood of a word given its preceding words. In practice however, the context window is limited to a fixed number. Increasing the number allows more context but quickly becomes impractical for longer sequences. Neural models brought major advances to spelling correction. Instead of relying on

a limited context window, they can use the full sequence context. Hertel [5] framed spelling correction as a translation task, "translating" an erroneous sequence into a correct one using an LSTM-based sequence-to-sequence model. Walter [6] improved this approach with a Transformer model, achieving higher accuracy and adding the ability to correct whitespace errors. Today, large language models (LLMs) such as GPT represent the state of the art, showing strong performance in both spelling and grammar correction [7].

2.2 On-Device Spelling Correction

These developments have reached mobile keyboards. Commercial systems such as Google's Gboard and Microsoft's SwiftKey initially relied on n-gram models but now use neural networks. However, architectural details are rarely disclosed. Academic research offers more transparency. Ghosh et al. [8] presented an attention-based model for local deployment, focusing on character-level correction. Niranjan et al. [9] introduced a transformer-based model for autocorrection, also running on-device.

2.3 Server-Based Spelling Correction

Other approaches have explored server-side computation. Zhang et al. [10] presented a keyboard where users could retype a misspelled word at any cursor position, without deleting the original word. By pressing a button, an RNN running on a server determined which previously typed word should be replaced. Grammarly also follows a client-server architecture. Few details are disclosed beyond using an "AI model", but the need of an active internet connection indicates that corrections are processed on a server. However, Grammarly is technically not a mobile keyboard. It shows corrections in overlay bubbles and can be used as an extension for other

keyboards.

A recent development is Google's Proofread[11], a sentence- and paragraph-level correction feature integrated into Gboard. Proofread uses an LLM that runs in the Google Cloud. Several optimization techniques such as quantization, bucket inference and speculative decoding reduce the latency by 39.4%. Although the paper does not provide absolute latency values, a demo video shows a correction time of about three seconds for a three-sentence paragraph. Proofread has not yet been rolled out globally and is limited to a few selected devices in the United States, which prevents a direct comparison.

Most prior work has focused primarily on correction quality. Latency and user acceptance are rarely documented. A notable exception is the work of Lu [12], who implemented an n-gram based system and reported an average correction time of 69 ms. Although neither neural nor server-based, it provides a reference point for our evaluation.

Chapter 3

Background

This chapter introduces the technical background relevant to this thesis. We define spelling correction and explain the principles of client-server architectures, latency, and neural models.

3.1 Spelling Correction

Spelling correction is the process of identifying and correcting misspellings in text. Mobile keyboards typically integrate a suggestion bar above the character keys, which shows possible corrections to the user. We can distinguish between two main types of corrections: Word-level correction provides alternatives for individual words within a sentence. For example, in the sentence "Whats your naem?" word-level correction would suggest "What's" and "name" as corrections. Sentence-level correction would suggest an entire sentence such as "What's your name?".

3.2 Client-Server Architecture

In a client-server architecture, computational tasks are divided between client and server. The client sends requests to the server. The server processes them and returns the results. This approach differs from local processing, where all computations are performed on the client device itself. By using the server's hardware, computations can be performed faster and more efficiently. This reduces the client's processing load, memory usage and battery consumption.

However, the client-server architecture also introduces two main challenges. First, dependency on a stable network connection. Without network connection the client cannot retrieve corrections from the server. The second limitation is added latency, which is discussed in the following section. Sending text over network adds latency. If users need to wait too long for corrections to arrive, it may disturb the writing flow and acceptance of the system.

3.3 Latency

Latency refers to the end-to-end time between a user action and the system's response. In the context of mobile spelling correction, it is the delay between the user's text input and the display of correction suggestions. Latency is usually measured in milliseconds.

In a client-server architecture, the total latency can be divided into three components:

Client Processing Time (CPT): The time the client takes to process the input and send the request to the server. This includes capturing the text input, constructing the HTTP request and displaying the returned results.

Round-Trip Time (RTT): This represents the time the request takes to

travel across the network. It depends on factors such as connection type (e.g., Wi-Fi vs. mobile network) and the server location.

Server Processing Time (SPT): This includes the time of the server-side processing, including model inference, and any pre- or post-processing.

The total latency is given by:

$$\text{Latency} = \text{CPT} + \text{RTT} + \text{SPT} \tag{3.1}$$

3.4 Latency Requirements for Mobile Text Input

Guidelines from human-computer interaction (HCI) research provide general thresholds for acceptable response times. Nielsen [13] suggests that response times below 100 milliseconds are perceived as instant, while delays up to 1 second do not disturb the user's flow of thought. These values are widely used in user interaction research but are only general guidelines, not specific requirements.

For example, a 100 millisecond delay may be acceptable when opening a menu, but would be unacceptable for keystroke feedback, where users expect immediate character display. Therefore, the acceptable latency does not only depend on the duration, but also on the type of feedback and how it is integrated into the writing flow.

This dependency is particularly interesting for spelling correction systems as it suggests that there can be differences depending on the correction method. Word-level corrections that are triggered while typing may be more sensitive to latency. Even short delays may disrupt the writing flow. Sentence-level corrections which are triggered after the completion of a sentence may tolerate a higher latency, since pauses

often occur naturally after completing a sentence.

To our knowledge, there is no published research that investigates how latency affects user acceptance of different spelling correction modes.

3.5 Transformer Model

The Transformer [14] is a sequence-to-sequence neural model architecture that replaces recurrent layers with attention mechanisms. Unlike recurrent models, it processes all tokens of a sequence in parallel and captures relations between any positions through self-attention. This work employs a Transformer model consisting of an encoder and a decoder. The encoder processes the input sequence into a contextual vector representation. The decoder generates the corrected sequence token by token, using the encoder output and the previously generated tokens. The final output is a sequence of tokens, that is converted back to text sequence.

3.6 Beam Search

Beam search is a decoding algorithm that generates multiple candidate outputs from a neural model. At each step, it keeps the top k partial sequences with the highest cumulative probability. These are expanded by one token, scored again, and reduced back to the top k. This allows the model to produce several candidate sequences instead of returning only a single sequence as in greedy search.

Chapter 4

System Architecture

This chapter describes the architecture and implementation of the client and server components. The spelling correction system consists of two main components that are connected via HTTP:

- **The client Android keyboard**, which captures the user input, determines when a correction request should be sent and displays the suggestions from the server.
- **The correction server** which hosts the neural Transformer model and processes the correction requests.

4.1 Server Implementation

4.1.1 Server Architecture

The server is responsible for receiving incoming requests, running the neural model, post-processing its output, and returning the results to the client. It is implemented in Python and uses the Flask framework, which offers a simple API for HTTP requests.

Each request follows this pipeline:

1. **Input handling:** The server receives the HTTP POST request.
2. **Tokenization:** The received text is tokenized and encoded.
3. **Inference:** The encoded tokens are passed to the model, which produces the correct token sequence.
4. **Decoding:** The token sequence is converted back into readable text.
5. **Post-processing:** The corrected sequence is aligned with the original input to extract differences.
    - In **word correction mode**, the differences are used to show correction candidates for specific words.
    - In **sentence correction mode**, the differences are used to highlight which words were changed.

4.1.2 Neural Correction Model

As correction model serves a self-trained transformer-based sequence-to-sequence model, which follows the standard encoder-decoder architecture. The encoder maps the input to contextual representations, which the decoder uses to generate the corrected output sequence. This model is used for both word correction and sentence correction.

To prepare the input for the model, we use a subword tokenizer based on Byte-Pair-Encoding (BPE) with a vocabulary size of 8000. Subword tokenization splits words into smaller units, allowing the model to handle rare or unknown words more effectively than word tokenization. It also avoids long sequences produced by character tokenization.

The model was trained on a subset of the English Reddit dataset, which contains over 64 million sentences. The dataset was chosen for its conversational tone, typical for mobile text input. The model supports both uppercase and lowercase as well as common special characters. For training, we artificially introduced spelling errors

by inserting, deleting and swapping characters.

In mobile keyboards, corrections are often triggered mid-sentence while the user is typing. In these cases, the model may receive an incomplete sentence. A neural model that is only trained on complete sentences develops a bias toward generating complete sentences.

Input: “I can’t find teh”

Expected correction: “I can’t find the”

Model output: “I can’t find tea.”

In this example, although “teh” is an obvious typo for “the”, the model incorrectly replaces it with “tea” to complete the sentence. To avoid this, the model was not only trained on complete sentences, but also on sentence fragments.

An overview of the training configuration is provided in Table 4.1.

Before deployment, the trained model was exported to ONNX format to improve the inference speed. The model generates up to 35 tokens per sequence, which is sufficient for most sentences.

4.1.3 Postprocessing

The model generates corrected text sequences as output. On their own, the raw output cannot be used directly by the client. For sentence-level correction, the keyboard must know which words were changed to highlight them. For word-level correction it must receive explicit corrections for the current word. This post-processing step takes place after model inference.

For each correction request, the model performs beam search with a width of $k = 3$. This produces three corrected sentences. Each word from the original input sentence may therefore receive up to three alternative suggestions. To extract them, the server aligns each corrected sentence with the original input using the Python library `difflib.SequenceMatcher`, which detects insertions, deletions, and replacements

Parameter	**Value**
Architecture	Transformer Encoder-Decoder
Encoder Layers	2
Decoder Layers	2
Attention Heads	5
Hidden Dimension	256
Embedding Dimension	256
Total Parameters	12,533,312
Tokenizer	BPE (Byte-Pair Encoding)
Vocabulary Size	8,000
Max Sequence Length	35 tokens
Dataset	English Reddit (subset)
Training Sentences	64 million
Batch Size	256
Learning Rate	0.001
Optimizer	Adam
Epochs	2
Deployment	ONNX format
Beam Search Width	$k = 3$

Table 4.1: Model training configuration

at the word level.

For example, given the input “I can’t find teh file”, beam search may return the following sentences:

1. I can’t find the file
2. I can’t find that file
3. I can’t find these file

The alignment finds that at one position all three candidates differ from the original input. “teh” is replaced by “the”, “that” or “these”.

In word-level correction mode, these differences are aggregated into a list of suggestions for each word position. In sentence correction mode, the results are stored as list of differences that mark which words were changed in each sentence.

The results for both word- and sentence correction are returned to the client in the JSON format as shown in the example below:

```
{
  "wordCorrections": {
    "3": { "original": "teh", "suggestions": ["the", "that", "these"] }
  },
  "sentenceAlternatives": [
    { "text": "I can't find the file",
      "differences": [{ "orig": "teh", "corr": "the", "pos": 3 }] },
    { "text": "I can't find that file",
      "differences": [{ "orig": "teh", "corr": "that", "pos": 3 }] },
    { "text": "I can't find these file",
      "differences": [{ "orig": "teh", "corr": "these", "pos": 3 }] }
  ]
}
```

At first sight, returning both word and sentence corrections may appear redundant. However, the alignment is already computed on the server and can be reused for both correction modes. This avoids any computation on the client to keep it as minimal as possible.

The complexity of the alignment is $O(k \cdot n^2)$, where n is the sentence length and k the beam width. Given the length of typical sentences, the cost is negligible compared to model inference.

4.2 Client Implementation

This section describes the implementation of the Android keyboard and its interaction with the correction system. The keyboard captures the user input, sends it to the server and displays correction suggestions.

4.2.1 Keyboard Architecture

The keyboard app is written in Kotlin and builds on the open-source project *BasicKeyboard*[1]. BasicKeyboard provides a minimal keyboard implementation without any additional features, which makes it a clean starting point for further development. The total size of the app is 13 MB.

The app is implemented as Input Method Editor (IME), which allows it to be used system-wide in all Android applications. To activate the keyboard, users must enable it in the Android settings and set it as default input method.

The keyboard is visually divided into two areas: The lower area contains the standard character keys, arranged in a QWERTZ layout. Directly above the key area, a suggestion bar was added as part of this thesis. The suggestion bar is the main interface for users to interact with correction suggestions.

4.2.2 Correction Modes

The keyboard supports two modes of corrections: word corrections and sentence corrections. Both use the suggestion bar to display the results. Depending on the correction mode the bar either shows word suggestions or sentence suggestions. Suggestions can be accepted by tapping on them, which replaces the corresponding word or sentence.

Word corrections are triggered on every character input or when the user moves the cursor to a previously written word. In both cases, the keyboard extracts the current sentence from the text field and sends it to the server. The server returns up to three suggestions for the current word, sorted by likelihood. The suggestions are displayed so that the most likely correction appears in the center of the bar, the second most likely on the left and the third on the right.

[1] `https://github.com/modularizer/BasicKeyboard`

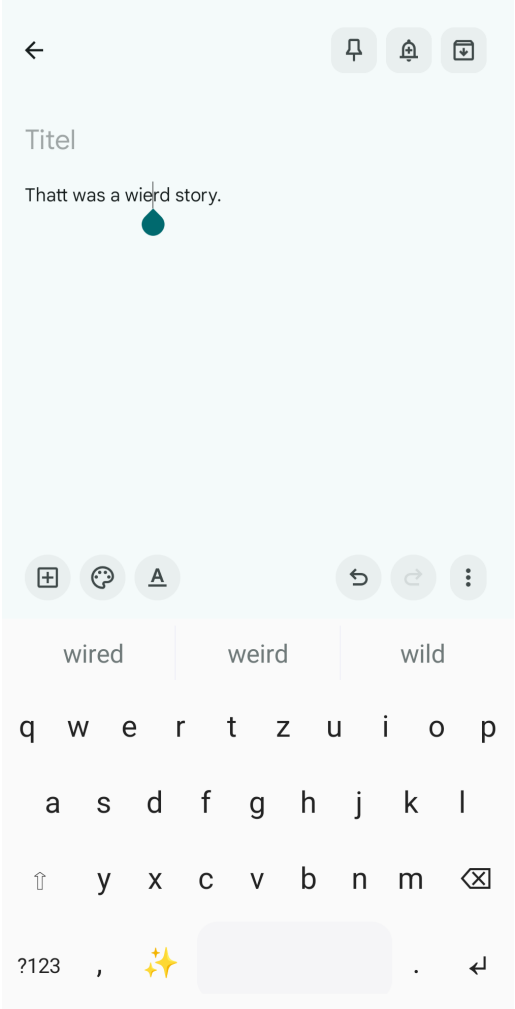

Figure 4.1: Word correction suggestions after placing the cursor on a word.

Figure 4.1 shows an example where the cursor is placed on “wierd”, and the server returns the suggestions “weird”, “wired”, and “wild”.

Sentence corrections are triggered when the user types a sentence-ending character (., ?, !) or manually presses the Magic Key located to the left of the space bar. In both cases, the keyboard again extracts the current sentence from the text field and sends it to the server. The server returns three sentence suggestions. The first suggestion is displayed in the suggestion bar. By tapping the Magic Key, users can cycle through the other sentences and choose the ones they prefer. In each displayed sentence suggestion, modified words are highlighted to make it easier for users to

see what changed.

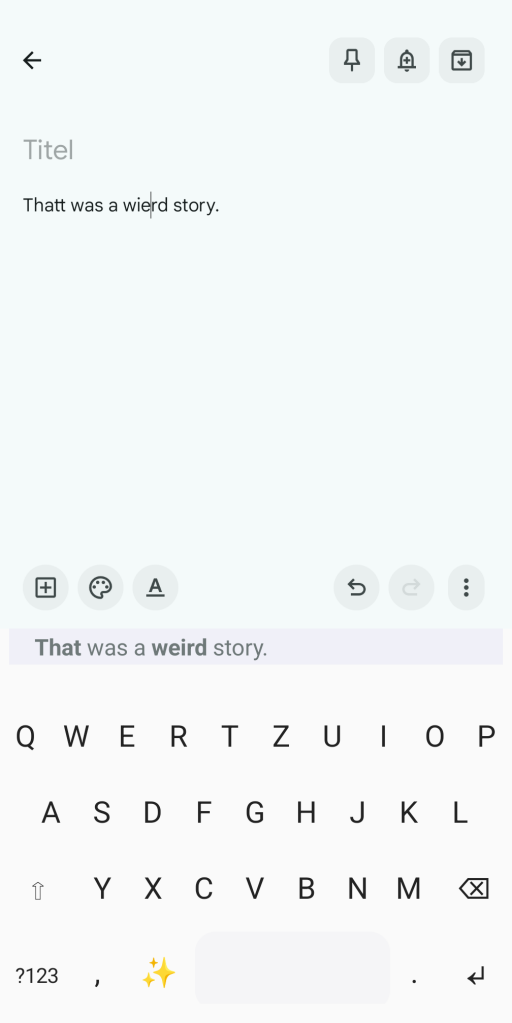

Figure 4.2: Sentence suggestion with highlighted changes.

In Figure 4.2 the original sentence ”Thatt was a wierd story.” is corrected to ”That was a weird story.”, with ”the” and ”weird” highlighted to show the change.

4.2.3 Correction Behavior

All corrections require an explicit confirmation from the user. The keyboard never replaces text automatically. This design ensures that users keep full control over their text.

Automatic correction was intentionally discarded. Prior studies have shown that automatic word correction can slow users down, especially when incorrect suggestions must be undone[15]. For sentence corrections, automatic replacements may cause changes that happen far from the current cursor position. Unnoticed changes outside the user's focus can damage user trust. To prevent these effects, corrections are only applied when the user actively taps on a suggestion.

4.2.4 Interaction and Error Handling

The keyboard continuously monitors the connection to the correction server. Every 10 seconds, a ping is sent to server to check its availability. If the server does not respond within two seconds, the user is informed by the connectivity issue by a message "Offline" displayed on the space bar, as seen in 4.3.

In situations where the network is unstable or slow, corrections may arrive late. To prevent disruptions while writing, user can manually disable the correction feature. A long press on the space bar toggles between correction enabled and disabled. When disabled, no data is sent to the server and the suggestion bar remains hidden.

This option can also be useful in situations where users prefer not to send text to the server. Although all communication is encrypted using HTTPS, the correction on the server is done on unencrypted text. Temporarily disabling corrections gives the users control over which data is transmitted.

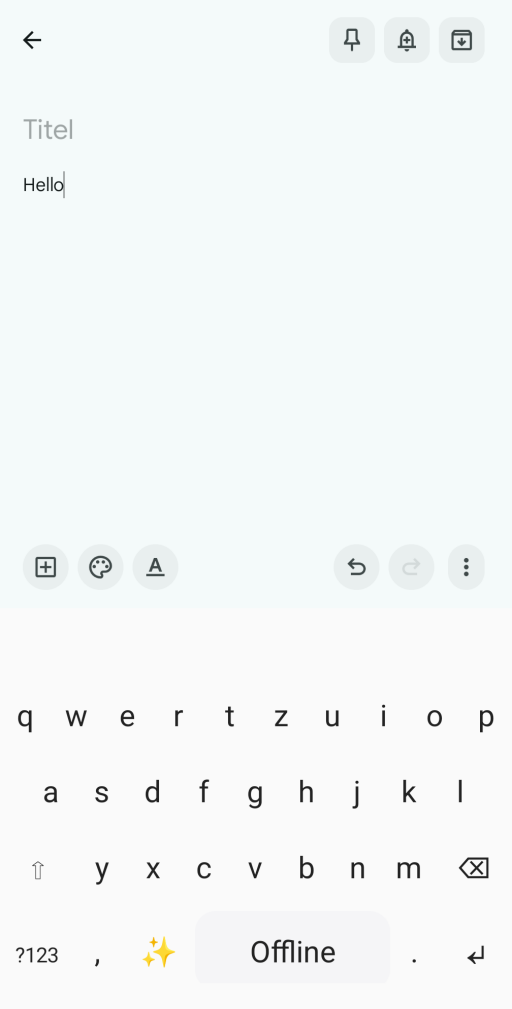

Figure 4.3: The space bar shows ”Offline” to indicate that no suggestions are currently available due to missing server connection.

Chapter 5

Evaluation

This chapter presents the evaluation of the system. It consists of a technical evaluation and a user study. The technical evaluation analyzes the latency of the client-server architecture and compares the correction quality with Gboard, a keyboard that uses a neural network. The user study complements this by assessing how people accept the system in practice.

5.1 Technical Evaluation

5.1.1 Setup

For the technical evaluation the system was tested under realistic usage conditions. The correction server runs on a university-hosted machine equipped with an NVIDIA RTX 4090 GPU. Unlike laboratory setups, the server is not located in the same local network as the client. Instead, the client communicates with the university network through a VPN tunnel. As client device, we used a Google Pixel 9 Pro XL smartphone running Android 16.

For the evaluation, we sampled 500 random sentences from the Reddit dataset. It is the same dataset from which we sampled the training data of the model, but

from a split that was not used in training. Artificial spelling errors were introduced using the same character level edits as in the training phase, including character swaps, deletions, and insertions. Different from training, we now also included a list of frequent misspellings including errors such as homophones (e. g. there → their) for more realistic and diverse error types. For some measurements all 500 sentences were used, for others only a smaller subset.

5.1.2 Latency

Before evaluating the latency of the entire system, we measured the inference time of the correction model in isolation on 500 sentences. As a reference point, we included results for k=1, which corresponds to greedy decoding. Since the keyboard requires three corrections in the suggestion bar, the regular system runs with k=3. Table 5.1 shows the average inference times for different beam sizes measured on our test dataset.

Beam Size	Mean (ms)	Std (ms)
1	13.92	3.40
3	41.75	10.73

Table 5.1: Model inference time per sentence

Beam search approximately triples the runtime compared to greedy decoding. With $k = 3$ the model completes correction in about 42 ms.

We then investigated how the inference time of the model scales with sentence length. This is relevant as the inference time should not grow disproportionally with longer inputs. We use the dataset with 500 sentences. Each sentence was processed incrementally, starting with the first word and extending one word at a time until the full sentence was reached. Results for both $k = 1$ and $k = 3$ are shown in Table 5.2. The

runtime increases almost linearly with the sentence length. With $k = 1$ inference time increases by about 1 ms per word and about 3 ms per word with $k = 3$.

Length	Mean k=1 (ms)	Mean k=3 (ms)
1	2.34	7.27
2	3.62	9.24
3	4.72	12.89
4	5.78	16.15
5	6.84	19.55
6	7.96	23.02
7	9.15	26.84
8	10.26	30.40
9	11.34	33.90
10	12.40	37.26
11	13.49	40.74
12	14.68	44.63
13	15.87	48.46
14	16.98	52.81
15	17.96	55.11

Table 5.2: Runtime by sentence length

After measuring the model in isolation, we evaluated the latency of the entire system on a subset of 50 sentences. The total latency includes all steps, from sending the request to displaying the suggestions on the device. The keyboard logged the total latency, while the server logged the server processing time (SPT). By subtracting SPT from the total latency, we get the network latency (RTT). Table 5.3 shows the average times with beam search k=3.

Component	Mean (ms)
RTT (network latency)	83.28
Server processing (SPT)	44.97
- Model inference	44.70
- Post-processing	0.26
End-to-end total	128.25

Table 5.3: End-to-end latency breakdown (50 sentences)

Network latency dominates in the total latency with 83.23 ms, while postprocessing is almost negligible. Corrections for full sentences appear on average after 128.25 ms. For reference, Lu [12] reported an average time of 69ms for a local n-gram-based model without network communication. In our system, model inference takes an average of 45.70 ms.

5.1.3 Accuracy

In addition to latency, we compared the correction accuracy of our keyboard with Gboard, a keyboard that uses a local neural network. The evaluation was performed manually on the reduced dataset with 50 sentences. Each sentence was copied at once into a text field and we checked which errors were corrected by each keyboard. We measured correction accuracy at the word level, based on the percentage of misspelled words that were corrected. We evaluated the Top-1 and Top-3 suggestions shown in the suggestion bar. The middle suggestion was considered the most likely one. To avoid adding context to the sequence, we did not apply any suggestions and kept the sentence unchanged.

For our keyboard, both word-level and sentence-level correction had to be considered. By default, we used word suggestions. However, our keyboard struggles with words that are concatenated. In these cases, the neural model often produces a correct sentence, but the extraction of the word suggestions can be inaccurate. When word suggestions could not correct the error, we additionally considered the sentence suggestions.

We counted all successfully corrected words from the total of 111 misspellings present in the dataset. Table 5.4 summarizes the results. Gboard achieved a Top-1 accuracy of 57.7% and a Top-3 accuracy of 66.6%, while our keyboard reached 83.8% in Top-1 and 89.2% in Top-3 correction accuracy.

While Gboard uses neural models, specific correction mechanisms are not disclosed.

	Top-1	Top-3
Gboard	57.7%	66.6%
Our keyboard	83.8%	89.2%

Table 5.4: Spelling correction accuracy comparison

Our significantly higher accuracy suggests that our model benefits from processing the full sentence context.

5.2 User Study

In this section, we present the user study which complements the technical evaluation. The measurements from the previous section describe the performance, but they do not show how they affect the user experience. Since no research offers a baseline for an acceptable latency on mobile spelling correction, it is unclear how the technical performance translates into usability. The user study therefore investigates how users perceive the system. In two modes we assess whether a client-server architecture can be considered feasible in a real-world scenario. The following sections describe the design, setup, procedure of the study, followed by an analysis of the results.

5.2.1 Methodology

Design

The study focused on user feedback of two separate correction modes: word correction and sentence correction. These two modes were not designed as competing approaches but as alternative use cases of the same client-server architecture. The intention was to compare different correction granularities and to examine if at least one of them makes the architecture feasible in practice.

Participants

Seven participants took part in the study. Their ages ranged from 20 to 27 years, with four male and three female participants. On their personal phones, all participants use the preinstalled system keyboard: four used Apple's keyboard, three used Google's Gboard.

Procedure

The study used the same Android device and server setup that was used in the technical evaluation. Each participant completed the study individually. At the beginning, each participant received an introduction to the tasks. The participants were then given one minute to get comfortable with the keyboard layout and the correction methods.

The study consisted of two phases. In each phase, the participants were asked to transcribe ten predefined sentences. The sentences contained no artificial errors, since a pilot test showed that participants naturally introduced enough typing errors to correct.

In the first phase, only word corrections were available. Participants could accept suggestions directly or afterwards, by placing the cursor on the respective word. If no suitable correction was available, they were asked to correct the error manually.

In the second phase, only sentence corrections were available. Participants were free to apply sentence corrections while typing or after finishing the sentence. If none of the three sentence suggestions fully corrected the sentence, they were asked to choose the most suitable and correct the remaining errors manually.

To avoid order effects, four participants started with word-level correction, followed by sentence-level correction. Three participants started in the opposite order.

After each phase, the participants answered a questionnaire consisting of seven questions. The questions addressed comfort, typing speed, correction quality, cognitive

effort, correction speed, trade-off between accuracy and latency, and overall satisfaction. The answers were recorded on a five-point Likert scale. The Likert scale enables to conversion of responses into numerical values, for a quantitative analysis. At the end the participants answered four additional questions, regarding their intention of using the systems daily and possible privacy concerns. A final open question asked about potential reasons against an everyday use.

5.2.2 Results

Word Correction Mode

Question	Mean	SD
Q1: Typing in this mode was comfortable.	3.43	1.13
Q2: I was able to type the sentences quickly.	2.86	1.07
Q3: The corrections were helpful.	3.57	0.98
Q4: I often had to think about whether I should accept a correction. (inverse)	2.43	0.53
Q5: The corrections appeared fast enough to be useful.	3.71	0.95
Q6: I would prefer more accurate corrections, even if they take slightly longer.	2.57	0.79
Q7: Overall, I am satisfied with this mode.	3.29	0.95

Table 5.5: User study results for Word Correction

Word correction achieved moderately positive feedback. Participants found the corrections helpful (Q3: 3.57) and generally appeared fast enough (Q5: 3.71). Comfort of use was rated slightly positive (Q1, 3.43), while typing speed was rated below neutral (Q2: 2.86). Question 6 (2.57) suggests that users are sensitive to delays, even when accuracy is acceptable. Overall satisfaction was moderate (Q7: 3.29), indicating that the system worked with word correction, but did not fully convince

participants. Table 5.5 presents the results of word correction mode.

Sentence Correction Mode

Table 5.6 summarizes the results for the sentence-level mode.

Question	Mean	SD
Q8: Typing in this mode was comfortable.	4.14	0.90
Q9: I was able to type the sentences quickly.	3.71	1.38
Q10: The corrections were helpful.	4.29	0.95
Q11: I often had to think whether I should accept a correction. (inverse)	3.00	1.00
Q12: The corrections appeared fast enough to be useful.	4.14	1.21
Q13: I would prefer more accurate corrections, even if they take slightly longer.	3.57	1.27
Q14: Overall, I am satisfied with this mode.	4.00	1.00

Table 5.6: User study results for Sentence Correction

Sentence correction achieved consistently better feedback than word correction, particularly, in comfort (Q8: 4.14) and the quality of corrections (Q10: 4.29). Overall satisfaction was clearly positive (Q14: 4.00). Typing speed (Q9: 3.71) and correction latency (Q12: 4.14) were rated above neutral. Participants showed that they were willing to accept even more latency in return for higher accuracy (Q:13 3.57). Table 5.6 presents the result of sentence correction.

General Acceptance and Privacy

The next four questions, which addressed general acceptance and privacy concerns are summarized in Table 5.7. The results give a mixed picture about long term use. Both modes were seen as possible for daily use, with sentence rated slightly higher

(Q16: 3.71). The need for an internet connection (Q17: 2.57) and the processing of the text on a remote server (Q18: 2.14) were seen critically.

Question	Mean	SD
Q15: I would use mode A for daily use.	3.43	0.79
Q16: I would use mode B for daily use.	3.71	0.95
Q17: I would use this keyboard regularly even if it requires continuous internet connection.	2.57	0.79
Q18: I am comfortable with my typed text being processed on a remote server.	2.14	0.90

Table 5.7: General acceptance and privacy questions

The final question about potential reasons against an everyday use of the systems revealed that for several participants privacy was an exclusion criterion. Others pointed to usability and missing features like punctuation after double-space, automatic removal of spaces before punctuation, sliding on the spacebar to move the cursor, absence of familiar bubbles next to misspelled words and visual key feedback while typing. A third category of feedback expressed the desire of using a hybrid solution combining sentence corrections, word corrections and autocorrection. When asked whether latency would affect their decision to use the keyboard, all participants stated that the latency was largely unnoticed and acceptable.

5.2.3 Discussion

The results of the user study show that both correction modes worked in practice. Participants were able to type, correct and complete the tasks in both modes without major difficulties. Sentence correction was rated particularly positive. Even though the correction quality was already considered good, the participants stated that they would accept additional delays for better corrections. This suggests that additional latency does not necessarily conflict with usability.

The study shows that the architecture is successful from a user perspective. At the same time, it shows that acceptance does not depend on correction quality and speed alone. Features and privacy reasons were major reasons why some participants remained hesitant.

Chapter 6

Conclusion and Future Work

Server-based correction is technically feasible for mobile keyboards: In our evaluation we showed that a client-server architecture can deliver interactive response times, on average 128 ms per sentence.

Latency was not perceived as a major issue: Although the network latency dominates the total latency, participants did not perceive it as disruptive.

Model accuracy outperforms commercial keyboard: On our test set with 111 spelling errors our model achieved an 83.8% Top-1 and 89.2% Top-3 correction accuracy, clearly outperforming Gboard.

We think there are many interesting aspects that could be explored in future work:

- **User acceptance and latency thresholds:** Our user study indicates that sentence-level corrections are well accepted in our architecture. However, exact boundaries of acceptable latency remain unclear. Sentence-level correction may be more tolerant of higher latency. A study with a larger number of participants could provide more insights into what latency levels users find

acceptable.

- **Improving word extractions:** In some cases, the model generates a correct sentence, but on word-level the current extraction and keyboard integration are not able to correct the error. More robust alignment methods, especially for handling concatenated and split words, could resolve the issue.
- **Model optimization:** Inference speed could be further improved through model optimization techniques such as quantization. At the same time scaling to larger transformer models or even LLMs might significantly increase correction quality.
- **Extending features and usability:** In addition to spelling correction the system could be extended to integrate features such as word prediction. Together with improvements in the UI this could bring the system closer to acceptance levels of commercial keyboards.

Bibliography

[1] S. Ravi, "Efficient on-device models using neural projections," *arXiv:1708.00630*, 2019.

[2] F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Communications of the ACM*, 1964.

[3] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, 1966.

[4] E. Mays, F. J. Damerau, and R. L. Mercer, "Context based spelling correction," *Information Processing and Management*, 1991.

[5] M. Hertel, "Neural language models for spelling correction," M.S. thesis, University of Freiburg, 2019.

[6] S. Walter, "Transformers and graph neural networks for spell checking," M.S. thesis, University of Freiburg, 2022.

[7] G. T. C., "Leveraging large language models for spelling correction in turkish," *PeerJ Computer Science*, 2025.

[8] S. Ghosh and P. O. Kristensson, "Neural networks for text correction and completion in keyboard decoding," in *Proceedings of CHI*, 2015.

[9] A. Niranjan, M. A. B. Shaik, and K. Verma, "Hierarchical attention transformer architecture for syntactic spell correction," *CoRR*, 2020.

[10] M. R. Zhang, H. Wen, and J. O. Wobbrock, "Type, then correct: Intelligent text correction techniques for mobile text entry using neural networks," in *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, 2019.

[11] R. Liu, Y. Zhang, Y. Zhu, *et al.*, "Proofread: Fixes all errors with one tap," *arXiv:2406.04523*, 2024.

[12] Z. Lu, *Spelling correction and autocompletion for mobile devices*, 2021.

[13] J. Nielsen, *Usability Engineering*. Academic Press, 1993.

[14] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *arXiv:1706.03762*, 2017.

[15] O. Alharbi, W. Stuerzlinger, and F. Putze, "The effects of predictive features of mobile keyboards on text entry speed and errors," *Proc. ACM Hum.-Comput. Interact.*, 2020.