Augmented Reality Grammar Checker: A Study on Time Behavior and Educational Usability

Muhammad Arifin Hidayat
muhammadarifinhidayat@student.utwente.nl
University of Twente
Enschede, The Netherlands

Abstract

This study explores the implementation of large language models in an offline augmented reality grammar checker on resource-constrained devices, broadening accessibility for Indonesian learners. The primary goal is to provide rapid and accurate grammatical corrections for text captured by learners. Using sub-1 billion parameter models (Gemma3 1B, Olmo2 1B, Qwen3 0.6B, and Llama3.2 1B), I evaluated the performance impacts of 8-bit and 4-bit post-training quantization using the llama.cpp inference framework for Android.

Results show that 8-bit quantization maintains grammatical error correction accuracy within 0.02% of GLEU and 1.3% of the F0.5 score. Additionally, 4-bit quantization reduces GLEU by approximately 3.2% and the F0.5 score by 18.2% relative to float16. However, quantization enhances inference speed, improving prompt processing throughput by up to 167% and token generation performance by roughly 124%. Furthermore, hardware acceleration using the 4-bit Olmo2 model on Snapdragon 8 Gen 1 delivers up to an 8.5x speedup (172 tokens per second prompt processing and 8.28 tokens per second token generation) with 380.2 ms latency.

A user study of 18 Indonesian high school students resulted in a mean system usability scale score of 59.3 ± 15.1, indicating "OK" / "marginally acceptable" usability. Qualitative feedback pointed to a steep learning curve, reliance on assistance, perceptible latency, and limited device compatibility.

Taken together, this study demonstrates large language model optimizations for a fast, accurate, and usable offline augmented reality grammar checker on resource-constrained devices. However, this study is constrained by limitations in devices, frameworks, and corpus size, pointing to future work on broader device validation, different frameworks, and larger Indonesian error datasets.

Keywords: augmented reality, large language model, grammar correction, on-device inference, llm optimization

1 Introduction

Augmented reality (AR) technologies have transformed educational practices by creating more engaging and interactive learning experiences. Prior studies have found that student attention declines after 10–15 minutes during lectures. Attention span, though variable across learners, can be prolonged when the learning experience is highly engaging [7]. AR has been shown to improve grammar and reading comprehension, raise motivation, and increase engagement [20, 27]. Adapting AR for a grammar checker application can enhance language learning by offering grammatical feedback on physical text, such as printed materials or handwritten notes. A prior study explored a cloud-based grammar checker for printed documents and suggested extending it to handwritten text and AR integration [41]. However, existing grammar checkers and optical character recognition (OCR) tools often rely on server-based processing, which can introduce latency issues, raise privacy concerns, and limit accessibility in low-connectivity settings. Offline operation is essential for students in developing countries, such as Indonesia, where internet connectivity may be limited in several regions [1].

Large language models (LLMs) have the potential to enable on-device offline grammar checking. However, the time behavior and correction accuracy performance in a fully integrated offline AR environment remain underexplored, especially on resource-constrained devices. Moreover, resource constraints can degrade the performance of LLMs compared to server-side deployment. Consequently, optimizing small-scale LLMs (sub-1B parameters) is essential to mitigate on-device resource limitations. Some studies have used various LLM optimization techniques, such as quantization, knowledge distillation, hardware acceleration, and software acceleration, to address this challenge [17]. Although these techniques can optimize on-device performance, they often introduce trade-offs, particularly in accuracy, which is crucial for grammar checkers. Furthermore, LLM-based grammar checkers are rarely tailored to the grammatical error distributions of non-native English speakers, such as Indonesian learners, who frequently struggle with tense, noun, and article usage [49]. The Indonesian language has different tense and article rules (no tense conjugations and no articles), causing systematic first language (L1) transfer errors [34]. Some research emphasizes the value of specialized datasets and techniques in advancing grammatical error correction (GEC) systems, particularly for challenging error types, conversational contexts, and low-resource languages [30, 48]. For this reason, curating a dataset that reflects common mistakes

can improve the robustness of the LLM grammar checker in addressing learners' error patterns.

Time behavior is a critical aspect of AR applications, as delays degrade user experience and the effectiveness of the learning tool. A user-acceptance study of latency suggests acceptable latency levels are around 300 ms for tapping tasks [36]. For AR adaptations involving high-precision micro-instructions, users begin to notice latency differences at 69 ms, preferring systems with latencies of 132 ms or lower [16]. In AR musical collaboration, the minimum noticeable delay for audiovisual integration is between 160 and 320 ms, with delays becoming less tolerable at 320 ms but remaining somewhat acceptable even up to 1200 ms [18]. Therefore, achieving low latency is important for seamless user interaction in AR environments and maintaining learners' attention spans.

In this study, I fine-tuned sub-1B LLMs (Gemma3-1B, Olmo2-1B, Qwen3-0.6B, and Llama3.2-1B) using a tailored dataset based on Indonesian error distribution and proficiency level. I integrated the model in the full AR Grammar Checker pipeline from OCR, LLM, and AR. I also evaluated the trade-off of accuracy and time behavior of LLMs when optimized with 8-bit and 4-bit post-training quantization and explored further speed-up using hardware accelerators. Moreover, I benchmarked the LLM's inference in the llama.cpp framework on a mobile platform powered by Snapdragon 8 Gen 1, achieving up to 8.5x speedup (172 tokens per second prompt processing and 8.28 tokens per second token generation) with less than 1% degradation in GLEU and F0.5 scores compared to the 4-bit precision baseline without hardware acceleration. In the end, I evaluated the application usability for students in Indonesia.

The contributions of this work are summarized as follows:

- I developed a full pipeline of OCR, LLM, and AR in offline operation.
- I fine-tuned an OCR on the IAM handwriting dataset and evaluated its accuracy.
- I fine-tuned LLMs on 14944 sentence pairs from the public corpora (CoEdIT, NUCLE, FCE, W&I+LOCNESS) derived via error-distribution sampling from 1837 Indonesian learner sentences.
- I evaluated the time behavior and accuracy trade-off of the grammar checker using sub-1B parameter LLMs in 4/8-bit quantization and hardware acceleration.
- I evaluated the usability of the AR grammar checker with 18 Indonesian students in a classroom setting.

By bridging the gap between LLM optimization research and AR application constraints, this work provides a solution for deploying low-latency LLM grammar checkers in AR environments on resource-constrained devices. Furthermore, this study can serve as a guide for more on-device LLM deployment use cases in the future.

2 Related Work

2.1 AR Application in English Learning

AR has shown promising applications in English language learning. Studies have demonstrated AR's potential to enhance engagement, motivation, and learning outcomes for English as a Second Language (ESL) students [5, 26, 27, 45]. Moreover, AR technology offers unique advantages in creating immersive and interactive learning environments, potentially eliminating barriers in the English learning process for students of all ages [2, 12].

In practice, AR applications can support various aspects of language learning, including phonics, vocabulary, and basic listening and speaking skills [5, 12, 26]. Specific instructional strategies such as augmented word spelling games, card-based word visualization activities, and word annotations have been explored to make learning more engaging [2].

With recent advancements in artificial intelligence (AI), the integration of AR with LLMs opens new possibilities for more adaptive language learning systems. For instance, AI tools like ChatGPT have been shown to assist in preparing content for foreign language education [42].

Despite these potentials, little research has explored the integration of LLMs into offline AR applications for resource-constrained devices, where lightweight LLMs can fit the resource limitations.

2.2 On-device OCR

Some OCR technologies have enabled text extraction from various sources, including printed documents and handwriting. Convolutional neural networks (CNNs) have been used to recognize characters in Android environments, detecting Latin-based languages. However, this study was limited to a single character recognition and suggested using MLKit, which offers broader functionality [44].

Another widely used engine is Tesseract OCR, an open-source OCR engine based on Long Short-Term Memory (LSTM) networks. In comparative evaluations with cloud-based OCR solutions such as AWS Textract, Tesseract demonstrated competitive performance. For instance, when tested on a handwritten dataset, it achieved 76.4% accuracy, closely approaching Textract's 84.1% [29].

More recently, transformer-based models like TrOCR have pushed the state-of-the-art in OCR accuracy. TrOCR achieved character error rates (CER) below 0.05 on the IAM dataset [25, 28]. Despite their accuracy, transformer-based vision models are computationally intensive for on-device deployment.

These advancements in OCR technologies offer an opportunity to explore their capability to be integrated into an offline AR grammar checker application pipeline.

2.3 LLM Grammar Checker

Recent studies have explored the use of LLMs for grammatical correction in English. An LLM-based curriculum learning was introduced to improve GEC performance, demonstrating significant gains over baseline models and traditional curriculum learning methods. Furthermore, the results were closely aligned with human experts, indicating the reliability of LLMs on ESL learning [15].

Additionally, other research has noted that English language learners produce different writing errors than native speakers, with notable variation across proficiency levels. Given these differences, recent work has examined how LLM performance interacts with second language (L2) proficiency. Findings indicate that overcorrection occurs more frequently in texts written by advanced learners (level C) compared to beginner (A) and intermediate (B) levels [50].

In the context of Indonesian learners of English, studies have identified tense, noun usage, and article errors as among the most frequent error types. These patterns are influenced by differences between English and Indonesian grammar, such as the absence of articles and verb conjugation in the native language [34]. Recognizing these patterns is crucial for tailoring GEC systems to learners' actual needs. Taken together, prior work highlights both the potential and limitations of LLMs in GEC tasks. However, few studies explicitly consider L2 proficiency in the evaluation and optimization of LLM-based correction systems, especially in the Indonesian learner context.

2.4 Grammatical Error Correction Corpora and Evaluation

To train and evaluate a grammar error correction system, a suitable dataset must be used to gain the desired performance. Datasets can be obtained through manual collection or using available public datasets. Building GEC corpora from the ground up poses some challenges, such as annotation consistency, expensive quality control, and data collection [10]. As an alternative to addressing this challenge, numerous GEC corpora have been developed to support the construction of a GEC system.

The First Certificate in English (FCE) corpus is a subset of the Cambridge Learner Corpus, consisting of 1,244 answers to FCE exam questions by international L2 learners [46]. The National University of Singapore Corpus of Learner English (NUCLE) comprises 1,400 essays authored primarily by Asian undergraduates from the National University of Singapore. Another dataset, CoEdIT [35], introduces a combined dataset of various text editing tasks such as Lang-8, DiscoFuse, ParabankV2, and others to facilitate task-specific instruction tuning. Furthermore, W&I+LOCNESS [47] provides 3,600 annotated submissions of non-native English students with different Common European Framework of Reference for Languages (CEFR) levels: A (beginner), B (intermediate), and C (advanced).

In terms of GEC evaluation, there are some commonly used metrics that can be implemented to evaluate the performance of a GEC system. The grammatical ERRor ANnotation Toolkit (ERRANT) scorer measures performance by using edit-based F-score [9]. ERRANT uses the M2 format (Figure 1) to annotate parallel English sentences with error type information to provide robust error detection and correction. Moreover, some popular GEC corpora (FCE, NUCLE, Lang8, and W&I+LOCNESS) also have been standardized using ERRANT.

Original: This are gramamtical sentence .
Corrected: This is a grammatical sentence .
Output M2:
S This are gramamtical sentence .
A 1 2|||R:VERB:SVA|||is|||REQUIRED|||-NONE-|||0
A 2 2|||M:DET|||a|||REQUIRED|||-NONE-|||0
A 2 3|||R:SPELL|||grammatical|||REQUIRED|||-NONE-|||0
A -1 -1|||noop|||-NONE-|||REQUIRED|||-NONE-|||1

Figure 1. M2 format structure

Another metric, GLEU [31], is a sentence-based evaluation metric that does not rely on explicit edit annotations but only on corrected reference sentences. GLEU rewards n-grams in the system output that match the reference but not the original text, while penalizing n-grams that match the original text but not the reference.

Additionally, according to a recent study, LLMs themselves can act as automatic GEC evaluation. Kobayashi et al. [24] demonstrated that GPT-4 can achieve Kendall's rank correlation of 0.662 with human evaluations, surpassing other approaches such as meta-evaluation and machine translation evaluation.

Despite the availability of various corpora and evaluation metrics, developing effective GEC systems still faces challenges related to domain adaptability and reliability, especially in low-resource environments.

2.5 LLMs On-device Optimization

Deploying LLMs in a resource-constrained environment needs careful consideration to gain optimal performance. LLMs have to be small and fast so that the user can run them smoothly on their device. Jahangir et al. [17] explored various techniques to create faster and smaller models. They showed that model compression methods, such as quantization, could reduce model size and memory usage, resulting in faster inference speeds. However, these methods may introduce slight accuracy loss, particularly with aggressive compression techniques.

Quantization is a technique for reducing model size and computational requirements by lowering the precision of model weights and activations from high-precision floating-point formats to lower-precision formats like INT8 or FP16. This reduction in precision decreases memory usage and accelerates computations due to more efficient arithmetic operations [21]. Reducing the precision of model weights and activations to lower-bit representations significantly reduces computational load and accelerates inference. MobileBERT [38] showcased that knowledge distillation and quantization could create a model that is 5.5 times faster than the original BERT model with only minor accuracy degradation of less than 1% on the GLUE benchmark. This optimization achieved an inference latency of just 62 ms on a Snapdragon 855 phone. Another study, Z-FOLD [23], introduced a post-training quantization (PTQ) method tested on 128 random 2048-token segments from the C4 dataset that enhanced the performance of quantized models. By folding parameters and optimizing quantization schemes, Z-FOLD contributed to faster inference times while maintaining overall performance.

Leveraging hardware capabilities and optimizing software to apply specialized hardware accelerators like NPUs and GPUs can also significantly improve inference time [17]. However, these techniques depend on compatible hardware and may require adjustments to maintain accuracy. Some research has demonstrated significant speedup by offloading some layers to hardware accelerators. Panopoulos et al. [33] did a study using TensorFlow Lite's PTQ methods. They found that changing models to formats like FP16, DR8, and FX8 made inference 0.4–2.3 times faster. Moreover, they also demonstrated that employing accelerator delegates could achieve up to 18 times speedup after quantization. Accelerator delegates could transfer certain calculations to specialized hardware accelerators, resulting in major reductions in latency. However, some hardware may be incompatible with specific accelerators.

These findings highlight how quantization and hardware-specific optimizations can drastically reduce latency that requires immediate feedback to promote good user experience. However, a limitation of these techniques is their potential to lower model accuracy and compatibility issues, making it crucial to determine a balance between inference time and accuracy for an AR grammar checker, which requires rapid processing to provide immediate and accurate feedback within an AR environment.

By applying quantization, the grammar checker model becomes lightweight and efficient enough to run smoothly on mobile devices, ensuring a seamless user experience without severely compromising accuracy. While not all devices may support hardware accelerators, many modern smartphones include built-in NPUs, GPUs, or other accelerators that can significantly boost inference speed. Running LLMs in an accelerator allows the application to provide optimal performance on devices with hardware support while still functioning efficiently on devices without dedicated accelerators. This approach ensures broad accessibility while maximizing the user experience on capable hardware.

In this study, I explored the performance trade-off of a quantized sub-1 billion LLM grammar checker designed for Indonesian learners in an offline AR environment, and I evaluated how hardware acceleration affects the speed and accuracy of grammar checking.

2.6 Measuring Usability

Usability testing is critical for understanding how effectively users interact with the AR grammar checker application, giving insight into user satisfaction and application acceptance. A common approach to measuring usability is the System Usability Scale (SUS) [8]. The SUS consists of 10 statements about system usability:

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

The responses are rated on a five-point Likert scale from "strongly disagree" to "strongly agree." Items address perceptions of complexity, ease of use, confidence in using the system, and the necessity of technical support. SUS scores range from 0 to 100, with higher scores representing better usability. Table 1 illustrates how SUS scores can be categorized according to Bangor et al. [6].

Table 1. Adjective Ratings for SUS Scores

Adjective	Mean SUS Score
Worst Imaginable	12.5
Awful	20.3
Poor	35.7
OK	50.9
Good	71.4
Excellent	85.5
Best Imaginable	90.9

Achieving a high usability score should be targeted to ensure learners' ease of use. This research evaluated how high

school students use the AR grammar checker in a classroom setting to validate the application's usability.

3 Methodology

This section describes the methodologies used to evaluate the time behavior and usability of the AR grammar checker application. The planning process was created based on the Goals, Questions, and Metrics (GQM) framework formalized by Basili et al. [11]. The GQM approach aligned the evaluation with the research objectives and provided a structured methodology for data collection and analysis.

3.1 Goals, Questions, and Metrics

3.1.1 Goals. The primary goal of the experiment was defined as follows:

Analyze the performance of the AR grammar checker application.
For the purpose of optimizing time behavior.
With respect to latency.
From the point of view of end-users.
In the context of English language learning.

3.1.2 Questions. To achieve the defined goal, the following research questions were formulated:

RQ1: *What strategies can be implemented to improve the time behavior performance of an AR grammar checker application while maintaining accuracy?*

- **RQ1.1:** What is the effect of different quantization bits on the time behavior performance and accuracy of the LLM in the AR grammar checker?
- **RQ1.2:** In what ways does hardware acceleration further enhance the time behavior performance of the AR grammar checker?
- **RQ1.3:** How do combinations of quantization and hardware accelerators affect the overall latency and accuracy of the AR grammar checker?

RQ2: *How usable is an AR-based grammar checker for students in Indonesia?*

3.1.3 Metrics. The experiment measured the following key performance to answer the research questions:

Time Behavior Metrics Model load time, prompt processing time, token generation time, and overall latency.
Accuracy Metrics Correctness of the grammar suggestions.

3.2 OCR fine-tuning design

I explored two lightweight offline OCR libraries (MLKit [1] and Tesseract OCR [2]) with a size of ± 15 megabytes (MB), considering resource limitations and user accessibility. MLKit provides an uncustomized English OCR model and is more optimized for printed documents. Due to those limitations, I fine-tuned Tesseract OCR, which supports model fine-tuning, on a handwriting dataset. The dataset was obtained from the IAM database [28], which contained 10373 images of handwritten lines of text created by various writers. I combined the train and test split (9395 images) to obtain a larger training dataset for the model. The handwriting OCR model was trained using tesstrain [3] on a Windows laptop with an Intel i7 2.6 GHz CPU. The full training command was:

```
make training MODEL_NAME={model_name} \
START_MODEL=eng \
TESSDATA={path_to_tessdata} \
EPOCHS=10
```

3.3 LLM fine-tuning design

Figure 2 gives an overview of dataset and model preparation.

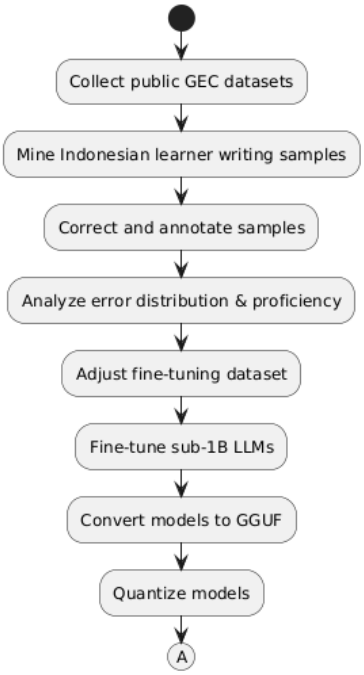


Figure 2. Dataset and Model Preparation

3.3.1 Dataset. I collected some publicly available GEC datasets: CoEdIT [35], NUCLE [13], FCE [46], and W&I+ LOCNESS [47]. These datasets were chosen because they cover writing samples from different sources across various English levels and are annotated by experts, providing dataset variety and annotation quality. However, none of these datasets show the common mistakes made by Indonesian learners of English, who often struggle with tense,

[1] https://developers.google.com/ml-kit/vision/text-recognition/v2
[2] https://github.com/adaptech-cz/Tesseract4Android
[3] https://github.com/tesseract-ocr/tesstrain

nouns, and articles because of their first language. To reweight the corpus toward Indonesian error proportions, I scraped over 1837 sample sentences from an online Indonesian forum [4] that discussed English writing improvement by its members. The data was then cleaned by removing duplicates, non-English text, and special characters. The cleaned data were corrected using the online grammar checker QuillBot[5] to obtain a parallel source-target pair. I aligned the edits to get error annotation using ERRANT [9], resulting in an M2 file whose tags quantify the frequency of error types. Figure 3 shows the error distribution for Indonesian learners.

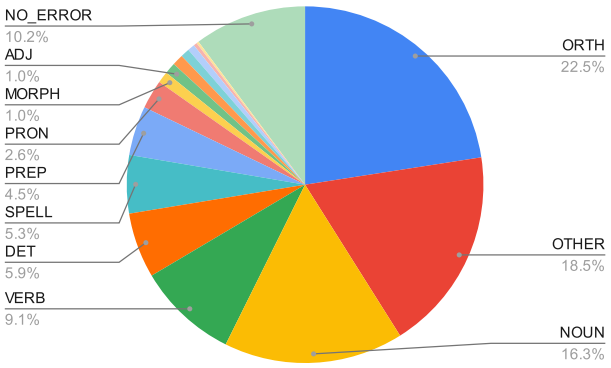


Figure 3. Indonesian Learner Error Distribution

Additionally, to adapt the dataset more toward beginner learners (CEFR A), I applied another filtering criterion based on the study of constructing the CEFR-based sentence profile [43]. CEFR A sentence could be categorized with the criteria of at least 70% A-level words and less than 15 words in length.

Using the combination of Indonesian error distribution and CEFR-level filtering, I filtered and gained a suitable dataset for training the LLMs that consisted of 14944 sentence pairs. This tailored corpus could mirror both the quantitative error profile and the proficiency level of Indonesian learners while preserving the annotation quality of established English GEC resources.

3.3.2 Model Selection. The offline Android AR grammar checker application will run the model entirely on-device, eliminating the need for cloud services. This solution ensures broader accessibility for Indonesian students, even in remote or underserved regions where internet connectivity may be limited [1, 37]. Due to resource limitations, I had to integrate the application with lightweight models. I selected four sub-1 billion models that were suitable for mobile deployment and fine-tuned them for the GEC tasks:

- **Gemma3 1B**: A state-of-the-art open model from Google that supports various frameworks [39].
- **Llama3.2 1B**: A multilingual LLM from Meta that has been trained on a broader collection of languages [14].
- **Olmo2 1B**: An open language model that has fully transparent training data, code, and recipe [32].
- **Qwen3 0.6B**: A latest model from Qwen that supports reasoning mode [40].

All of the models were decoder-only, selected based on compatibility with inference framework support (llama.cpp and MediaPipe). The latest models were selected to ensure the best performance on resource-constrained mobile devices.

3.3.3 Fine-tuning. LLM fine-tuning was done using the A40 GPU of the HPC Cluster at the University of Twente. I applied Low-Rank Adaptation (LoRA) to reduce the number of trainable parameters, resulting in faster and more memory-efficient training [19]. The number of LoRA-trainable parameters for each model:

- **Gemma3 1B**: 23.9M / 1.02B (2.3%)
- **Llama3.2 1B**: 90.2M / 1.33B (6.8%)
- **Olmo2 1B**: 96.5M / 1.58B (6.1%)
- **Qwen3 0.6B**: 80.7M / 677M (11.9%)

Table 2 describes the hyperparameters used for LoRA configuration. These configurations were selected because they provided a balance between fine-tuning efficiency and performance.

Table 2. LoRA Hyperparameter Settings

LoRA Parameter	Value	Description
Rank	128	Controls the number of low-rank factors
LoRA Alpha	256	Scaling factor for weight regulator
LoRA Dropout	0.1	Dropout rate to prevent overfitting
Target Modules	q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj	Modules to be fine-tuned

The model was fine-tuned using Hugging Face's Trainer API for 1 epoch to avoid overfitting. The training arguments for fine-tuning can be seen in Table 3.

To enable on-device inference across different platforms, I converted the fine-tuned model into a format compatible with the respective frameworks. Llama.cpp uses the GPT-Generated Unified Format (GGUF) format, which supports a wide range of post-training quantization levels, including float16 (f16), 8-bit, 4-bit, and even 2-bit quantization. In contrast, MediaPipe uses the Task Bundle format (.task),

[4] https://kask.us/gWzk9
[5] https://quillbot.com/grammar-check

Table 3. Fine-tuning Training Arguments

Training Argument	Value	Description
Batch size	4	Both training and evaluation per device
Gradient Accumulation Steps	4	Managing memory constarints
Epochs	1	Full cycle dataset training
Evaluation Steps	100	
Learning Rate	3e-5	
Weight Decay	0.1	Regularization
Warmup Ratio	0.1	Stabilize convergence
Precision	bf16	
Gradient Checkpointing	False	Avoid computation overhead

which currently supports only f16, 8-bit, and 4-bit quantized models. To ensure a fair and comprehensive performance evaluation, I included f16, 8-bit, and 4-bit variants in the study. This choice allows for a direct comparison of accuracy and inference efficiency across different quantization levels. However, when testing the on-device inference on MediaPipe, the converted models did not produce any output and returned empty results due to MediaPipe currently only supporting LoRA on GPUs for older models (Gemma 2B, Gemma2 2B, and Phi2). For that reason, I only tested the provided base model of 4-bit quantized Gemma3 1B for the MediaPipe framework.

3.4 System Design

Figure 4 illustrates the evaluation pipeline from designing the system to data analysis. After getting the fine-tuned models, I developed an AR application to test the model. The development of the AR grammar checker application required a robust and scalable software architecture to ensure maintainability and performance. The system was structured into modular components following the Model-View-ViewModel (MVVM) architectural pattern, which improves the separation of concerns and simplifies testing and maintenance [3].

The application was divided into the following key modules:

- **OCR Module:** Performs text recognition using MLKit or Tesseract OCR which offer on-device OCR capabilities optimized for mobile performance. For user testing, cloud OCR service was used to ensure OCR robustness.
- **LLM Module:** Integrates the optimized LLMs to provide grammar suggestions based on the recognized text. The llama.cpp[6] framework was used because it provides PTQ and hardware delegates. Another module was also created to evaluate MediaPipe[7]. Additionally, OpenNLP[8] was used to give error annotation based on the incorrect sentence and corrected sentence.
- **AR Module:** Uses SceneView[9] for easy integration of ARCore and Google's Filament rendering engine to overlay grammar suggestions onto the physical environment, providing an interactive AR experience.
- **Main Module:** Parent module that orchestrates the other modules.

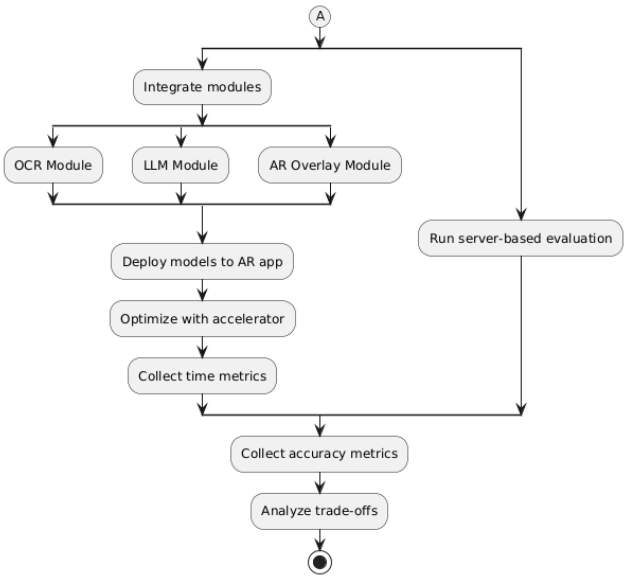


Figure 4. Evaluation Pipeline

Figure 5 shows the end-to-end workflow of the application. Appendix A.3 illustrates the user interface design of the application. Upon launching the app, users are prompted to select a grammar correction model from their local storage. The AR module concurrently activates the device camera to prepare for AR overlay, while the LLM module loads the selected model. Once initialization is complete, users can capture an image containing text. The captured image is processed by the OCR module, which extracts the textual content and returns it to the main module. The main module will show a popup dialog for OCR result checking. The checked text is then rendered as an overlay in the AR view.

Next, when the user initiates grammar checking, the main module sends the OCR-extracted text to the LLM module for inference. As the corrected output is generated token by

[6] https://github.com/ggml-org/llama.cpp
[7] https://github.com/google-ai-edge/mediapipe
[8] https://opennlp.apache.org
[9] https://github.com/SceneView/sceneview-android

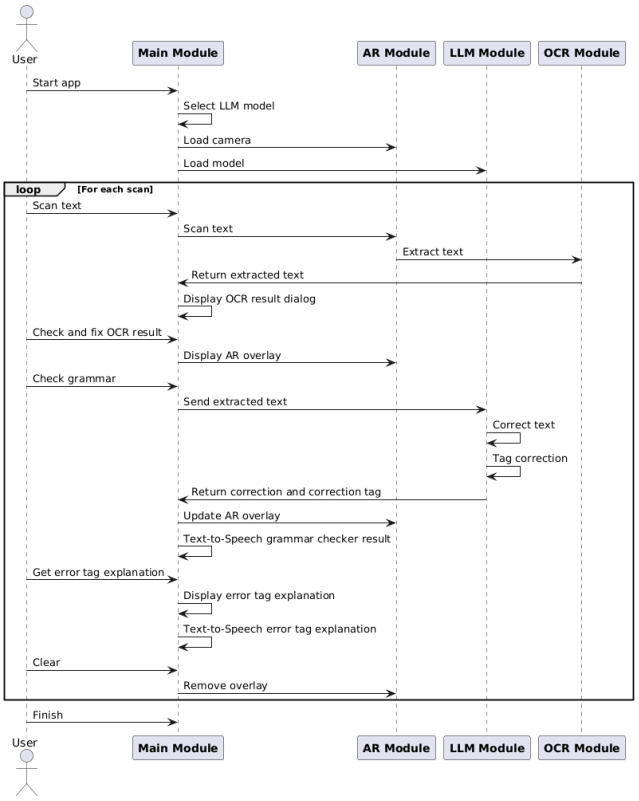


Figure 5. Application Flow Sequence Diagram

token, the AR overlay is updated dynamically in real time. Once the corrected sentence is complete, the system invokes a part-of-speech (POS) tagger to identify and annotate grammatical errors by comparing the original and corrected texts. The text-to-speech will speak the full results to the users. The main module parses these annotations and provides predefined explanations (Appendix A.4) for each identified error type. The text-to-speech feature will also speak the explanations. Finally, users may choose to reset the AR overlay and start a new OCR process, enabling iterative grammar checks within the same session.

3.5 Experimental Workflow

- **Optimization Techniques:** The experiment evaluated the impact of different optimization techniques, including:
  - **Quantization levels:** Models were evaluated at float16 (baseline), 8-bit, and 4-bit precision PTQ.
  - **Hardware acceleration:** OpenCL was used to offload computation to the GPU. Although MediaPipe officially supports acceleration, it is currently limited to a 4-bit Gemma3 model provided with fixed delegates. In contrast, llama.cpp provides both OpenCL and Vulkan delegates. However, Vulkan failed to support inference with input sequences longer than 32 tokens and delivered lower throughput than the CPU backend. As a result, OpenCL was the only usable delegate across models and devices for the llama.cpp framework.

3.6 LLM Workload

The LLM workload design was as follows:

- **Input Types:** LLM evaluation was conducted using two publicly available GEC datasets:
  - **W&I+LOCNESS:** Used for both ERRANT-based F0.5 scoring and LLM-based evaluation. This dataset (dev split) includes 1,037 annotated sentence pairs with CEFR-level metadata (A, B, and C), making it a standard benchmark for learner-oriented GEC tasks.
  - **JFLEG:** Used exclusively for GLEU evaluation. JFLEG contains 747 original sentences, each with four human-written references, and is widely used for evaluating fluency-preserving correction performance.
  - **Latency Input:** Module-level latency was measured with a fixed 15-word sentence, chosen to represent the upper bound of average sentence length for CEFR-A learners.
- **Inference Settings:** All LLM inference runs used a decoding temperature of 0.0 to minimize variability and ensure determinism across repeated trials.

3.7 OCR Workload

OCR evaluation used the validation split (976 images) of the IAM Line Handwriting dataset to test the OCR accuracy across all engines (MLKit and Tesseract OCR). Additionally, I also tried to evaluate the OCR performance qualitatively by capturing printed and handwritten sentences from different angles and light conditions.

3.8 Data Collection and Analysis

3.8.1 OCR Accuracy. The OCR evaluation used both quantitative and qualitative methods:

- **Quantitative:** OCR accuracy was measured using Word Error Rate (WER) and Character Error Rate (CER).
- **Qualitative:** OCR correctness was manually inspected under different angles and lighting conditions.

3.8.2 LLM Accuracy. Model output was evaluated against gold-standard references using three metrics:

- **GLEU:** Captures n-gram overlap, penalizing mismatches with original text.
- **F0.5 Score:** Computed using ERRANT, prioritizes precision over recall in grammar correction tasks.

- **LLM-based Evaluation:** A LLM score was obtained using the Llama3-70B model via the Groq API[10], simulating fluency and coherence-based human judgment. The evaluation was in a 10-sentence batch (Appendix A.2). This method builds on prior work showing high correlation between LLM-generated scores and expert annotation [24].

3.8.3 Latency and Speed. Time behavior metrics were collected using both the llama.bench utility (for prompt processing and token generation speed) and Android system logs (for end-to-end latency and model load time). Each configuration was run five times with at least 10-minute cooldown period between model runs to minimize the impact of thermal throttling.

Statistical Treatment: Descriptive statistics were reported across all optimization configurations. Accuracy–latency trade-offs were assessed qualitatively and numerically.

3.9 Environment and Procedure

3.9.1 Environment.

- **Server:** NVIDIA A40 GPU on the University of Twente HPC cluster, used for model fine-tuning and performance evaluation.
- **Device A:** OnePlus 10 Pro (Snapdragon 8 Gen 1, Adreno 730), used for all on-device performance tests.
- **Device B:** Pixel 4 (Snapdragon 855, Adreno 640), used for all on-device performance tests.

All on-device mobile evaluations were conducted offline, with Wi-Fi and mobile data disabled. Screen brightness was set to maximum, and background processes were minimized to reduce interference.

3.9.2 Testing Procedure. The evaluation process involved three phases:

1. **Few-shot Prompting Baseline:** Inference was first conducted using Hugging Face Transformers with 5-shot prompts to establish baseline performance (Appendix A.1).
2. **Full-Precision Model Evaluation:** LoRA fine-tuned models were evaluated in float16 precision directly after training.
3. **Quantized Model Evaluation:** Full-precision models were converted to GGUF format using llama.cpp, then quantized to 8-bit and 4-bit. Inference was conducted using llama.cpp on-device with and without OpenCL acceleration.

Application cache was cleared before each trial, and time behavior was measured within the full AR pipeline, from OCR trigger to overlay rendering, ensuring controlled environments. Additional timing logs were collected using logger codes in the application code to measure Time to First Token (TTFT) and model load time. For OCR accuracy testing, a different application was created to run all OCR engines, extracting the same test dataset.

[10]https://groq.com

3.10 User Testing Design

I conducted user testing with Indonesian high school students in a classroom setting to evaluate the usability of the AR grammar checker. Due to practical concerns around the OCR reliability of Tesseract OCR in unconstrained environments (student handwriting variability and poor lighting), cloud-based OCR was used in this phase to ensure consistent text extraction and prevent testing disruptions. The LLM module used 4-bit Qwen3 to make the testing preparation faster. Hardware acceleration was disabled to ensure the same environment for all users. This setup allowed students to interact with the full AR grammar checker as intended, while isolating potential OCR failures from influencing their user experience.

The testing involved 18 third-year high school students from SMA Kolombo Sleman. Students were grouped into 3 groups with a minimum of 1 working application per group due to compatibility. A worksheet was included for the students to guide them as they experienced the application (Appendix A.5). The first task was spotting error words in given sentences and using the application to verify their answers. The second task provided students with the opportunity to write five sentences describing their school holiday and evaluate their writing using the application. In the end, students filled out the system usability scale (SUS) questionnaire and gave some feedback for the application.

4 Result

4.1 OCR accuracy

As shown in Table 4, fine-tuning Tesseract OCR with a handwriting dataset reduced WER (Word Error Rate) from 0.793 to 0.2975 and CER (Character Error Rate) from 0.4072 to 0.1076 over the base Tesseract OCR English model (lower is better). Compared with the MLKit, the handwriting model achieved accuracy improvements in WER by 63.8% and CER by 77.0%. In qualitative evaluation, OCR results depended on how the image was captured (skewed angles and poor lighting resulted in degraded OCR accuracy).

Table 4. OCR Accuracy Across Different OCR Engines

Engine	WER	CER
MLKit	0.822	0.4683
Tesseract-ENG	0.793	0.4072
Tesseract-HTR	0.2975	0.1076

4.2 Quantization Effect

4.2.1 Accuracy. Table 5 presents the accuracy results of the evaluated LLMs under different PTQ implementations.

Table 5. Model Accuracy Scores: GLEU, F0.5, and LLM Evaluation

Metric	Qwen3				Gemma3				Llama3.2				Olmo2			
	few shot	f16	q8	q4	few shot	f16	q8	q4	few shot	f16	q8	q4	few shot	f16	q8	q4
GLEU	0.5792	0.8248	0.8247	0.8238	0.5851	0.8298	0.8298	0.8034	0.0588	0.8328	0.8340	0.8415	0.5468	0.8402	0.8413	0.8391
F0.5	0.1229	0.4328	0.4297	0.4238	0.0702	0.4416	0.4434	0.3612	0.0158	0.4663	0.4630	0.4599	0.0149	0.4805	0.4742	0.4643
LLM Eval	8.0800	7.2700	7.3100	7.1300	8.1200	7.8400	7.9200	7.8100	7.6300	7.6500	7.6700	7.3000	8.5200	7.2500	7.2400	7.0800

In general, quantization reduced a minor GLEU and LLM evaluation score. However, the F0.5 score showed a notable degradation. The GLEU score and LLM-based evaluation focus on fluency-oriented edits, meaning that lower values in these metrics may indicate reduced overall fluency or suboptimal grammatical restructuring. In contrast, the F0.5 score emphasizes precision over recall in edit operations, thereby favoring corrections that avoid unnecessary changes or overcorrection.

As shown in Figure 6, applying quantization led to a small accuracy reduction. Among the models, using 8-bit PTQ led to a negligible decrease in GLEU score, whereas larger accuracy degradations occurred when the bit precision was lowered to 4-bit. The highest reduction for 8-bit PTQ was observed in Qwen3, which had around a 0.01% accuracy drop. The Gemma3 model still had the same accuracy for GLEU. In another observation, 8-bit quantized Llama3.2 and Olmo2 models increased their accuracy by around 0.14% for the Llama3.2 model and 0.13% for the Olmo2 model.

With further quantization, compared to baseline f16, 4-bit PTQ resulted in greater accuracy degradation, decreasing by up to 3.18% for the Gemma3 model. At the same time, the other models' accuracy drops were less than 0.15%. In contrast, for 4-bit Llama3.2, the GLEU score further improved to 1.05%. Gemma3 was the most underperforming model in the GLEU score trade-off compared to the other models. The highest-performing 8-bit model was Olmo2 with a GLEU score of 0.8413, while the top-performing 4-bit model was Llama3.2, achieving a GLEU score of 0.8415.

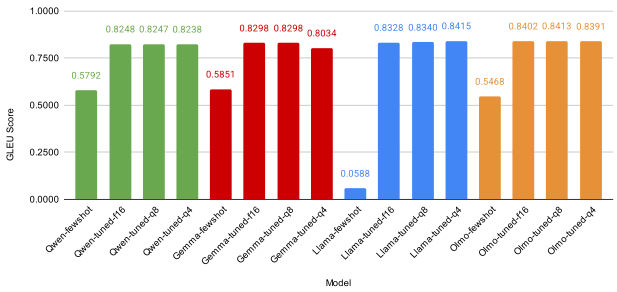


Figure 6. GLEU Score

Figure 7 shows that the F0.5 score was more sensitive to change than GLEU. The accuracy drops ranged from 0.71 to 18.21%, with 4-bit Gemma3 showing the largest drop. However, the 8-bit Gemma3 model behaved differently, increasing its accuracy by around 0.4%. Olmo2 achieved the highest GLEU and F0.5 scores among quantized models, whether at 8-bit or 4-bit PTQ. Among all models, Llama3.2 was the most stable after quantization, with F0.5 decreases of just 0.71–1.37% and GLEU gains of 0.14–1.04%, relative to f16.

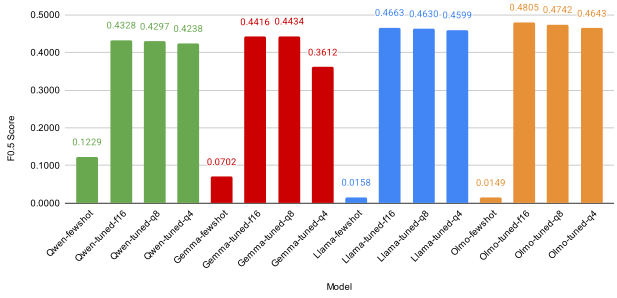


Figure 7. F0.5 Score

The LLM score evaluation from Llama3-70B showed different results for every model, as illustrated in Figure 8, varying from 7.08 to 7.92 out of 10 after quantization. Qwen3 gained more accuracy with 8-bit quantization, stating a 0.55% improvement, and then slightly reduced after 4-bit PTQ to a 1.93% accuracy reduction. A similar situation also happened with Gemma3 8-bit PTQ, where it obtained 1.02% accuracy, but it decreased 0.38% after 4-bit PTQ compared to the f16 precision model. Additionally, Llama3.2 also had accuracy improvement for its 8-bit model with a 0.26% accuracy gain, while it was the opposite for the 4-bit PTQ with a 4.5% loss. As the PTQ levels decreased, Olmo2 displayed a reduction in accuracy. While the same trend of increasing GLEU and F0.5 scores of the fine-tuned model compared to few-shot prompting was observed, a different result was noted from the Llama3-70B evaluation. Olmo2 had a gradual accuracy trade-off as the quantization went lower, but the others were in the same pattern: increased at 8-bit, then decreased at 4-bit compared to the base f16 model.

4.2.2 Time Behavior. Table 6 and Table 7 show the time behavior performance of each model, including inference time (prompt processing (pp) and token generation (tg)), measured in tokens per second (t/s), as well as model load time

Table 6. Model Inference Performance on Snapdragon 855: Prompt Processing (pp), Token Generation (tg), and Load Time (lt)

Metric	Qwen3			Gemma3			Llama3.2			Olmo2		
	f16	q8	q4	f16	q8	q4	f16	q8	q4	f16	q8	q4
pp (t/s)	19.33	44.40	42.00	10.74	33.00	29.60	7.52	21.60	20.20	7.44	20.80	19.40
tg (t/s)	11.67	18.60	19.20	6.82	12.80	13.40	5.48	10.98	12.40	5.36	11.16	12.20
lt (ms)	1021.33	566.60	534.40	1430.00	737.20	833.40	1459.20	1198.20	972.60	1604.20	896.00	689.40

Table 7. Model Inference Performance on Snapdragon 8 Gen 1: Prompt Processing (pp), Token Generation (tg), and Load Time (lt)

Metric	Qwen3				Gemma3				Llama3.2				Olmo2			
	f16	q8	q4	q4 accel	f16	q8	q4	q4 accel	f16	q8	q4	q4 accel	f16	q8	q4	q4 accel
pp (t/s)	22.40	44.00	42.40	144.00	13.92	31.80	26.00	142.00	9.40	25.00	20.80	112.00	8.44	22.60	20.60	172.00
tg (t/s)	11.80	18.40	20.20	6.74	6.84	12.80	12.00	16.00	6.32	13.80	12.18	8.50	5.70	12.80	12.60	8.28
lt (ms)	2657.20	2705.80	2523.80	4880.80	3128.00	2429.20	2487.00	5317.00	2754.20	2837.40	3088.80	5527.60	3270.40	2733.60	2592.40	5850.20

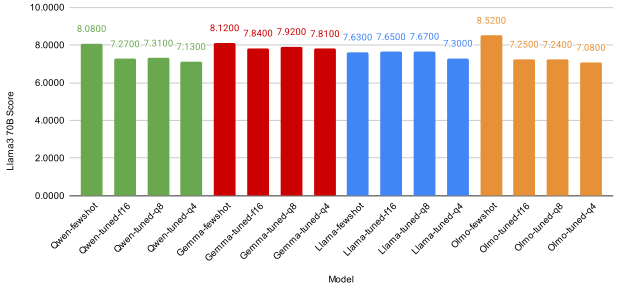


Figure 8. LLM Score

(lt), measured in milliseconds (ms). In summary, quantization enhanced the speed performance and reduced model load time. The results of inference time also showed that 8-bit precision models achieved more optimal inference performance than 4-bit PTQ.

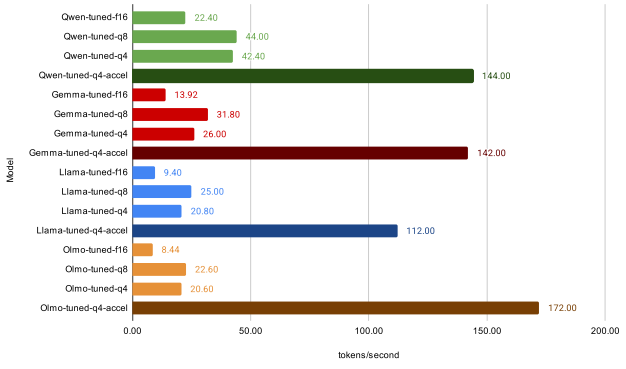


Figure 9. Prompt Processing on Snapdragon 8 Gen 1

Figure 9 and Figure 11 present, respectively, the prompt-processing and token-generation throughput obtained on a

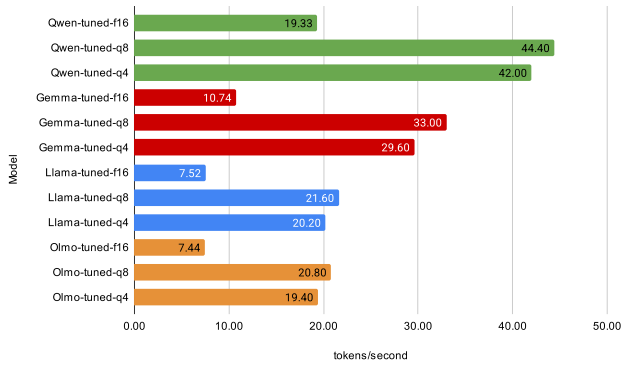


Figure 10. Prompt Processing on Snapdragon 855

Snapdragon 8 Gen 1, whereas Figure 10 and Figure 12 report the same metrics for a Snapdragon 855. Across both devices, prompt processing increased faster than token generation after quantization. All devices showed a notable speedup in 1B models, with an average base inference time of less than 10 t/s. After applying quantization, they achieved a speed of more than 10 t/s. For instance, the highest speedup was found at Olmo2, which was using 8-bit PTQ, achieving 167% prompt processing speedup and 124% token generation speedup on Snapdragon 8 Gen 1. On the lower CPU, Snapdragon 855, Gemma3 gained 3x prompt processing and 2.75x token generation speedup. Nevertheless, as Qwen3 had less than 1B parameters, its base prompt processing was already averaged at around 20 t/s and 10 t/s for token generation. After applying PTQ, Qwen3 gained the lowest speedup, peaking at almost double its speed on Snapdragon 8 Gen 1 and 2.3x on Snapdragon 855.

Using a lower quantization (4-bit) resulted in a decrease in prompt processing throughput for all models on both CPUs. Additionally, only Qwen3 experienced a token generation

speedup on Snapdragon 8 Gen 1, while the others experienced a speed reduction. Nevertheless, different results were found on Snapdragon 855, where all models gained faster token generation around 5–20 percentage points.

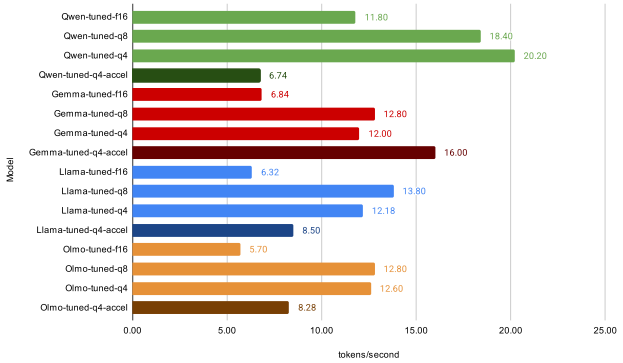


Figure 11. Token Generation on Snapdragon 8 Gen 1

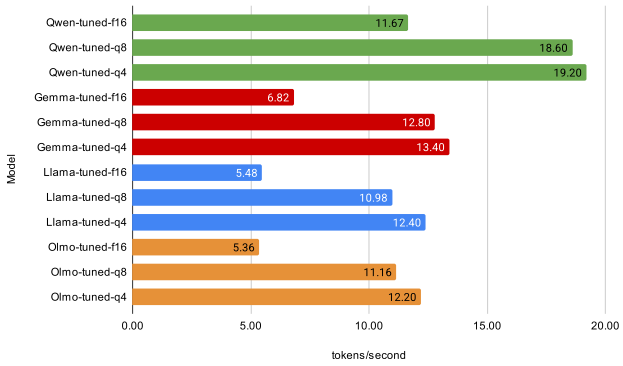


Figure 12. Token Generation Throughput on Snapdragon 855

In terms of model load time, Figure 13 and Figure 14, Snapdragon 855 required less load time compared to Snapdragon 8 Gen 1. In general, model load times were reduced for all models after quantization. 8-bit PTQ reduced load time on the Snapdragon 855 by 18–48% when compared to f16 precision. In contrast, only Olmo2 and Gemma3 gained a reduction in model load times on Snapdragon 8 Gen 1 by 16% and 22%, respectively. Llama3.2 and Qwen3 slightly increased their model load time by less than 3%.

Going further to 4-bit PTQ, 3 models (Qwen3, Llama3.2, and Olmo2) achieved a 3–15 percentage point reduction of 8-bit's load time, except for 4-bit Gemma3, which increased its load time by 7 percentage points on Snapdragon 855. However, on Snapdragon 8 Gen 1, Gemma3 and Llama3.2 raised 2 and 9 percentage points, respectively. Moreover, Olmo2 and Qwen3 reduced their percentage points by 4-7 compared to 8-bit PTQ.

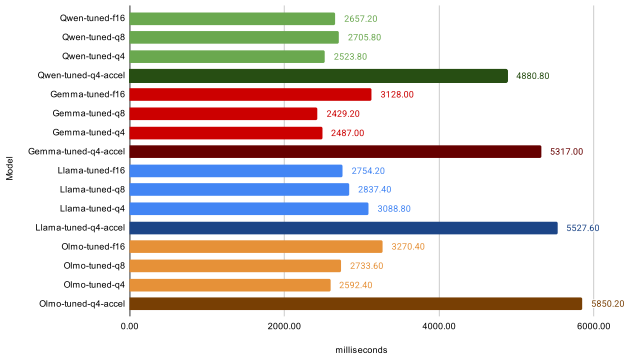


Figure 13. Model Load Time on Snapdragon 8 Gen 1

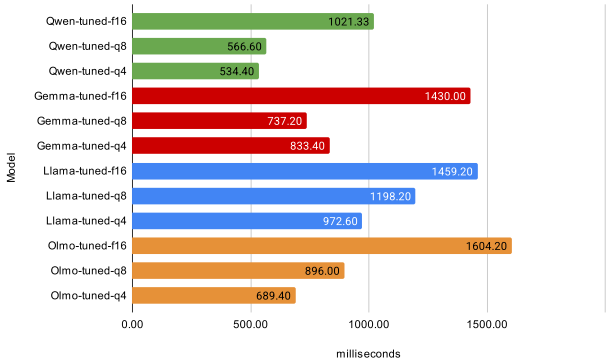


Figure 14. Model Load Time on Snapdragon 855

4.3 Hardware Acceleration Effect

The result of hardware acceleration speedup is displayed in Table 7. As a limitation of the testing device and selected framework, llama.cpp acceleration through OpenCL was only compatible with the OnePlus 10 Pro (Snapdragon 8 Gen 1) and 4-bit precision models. This investigation found that by offloading some model layers to a mobile GPU, it could significantly increase prompt processing by 6–20x compared to base f16 models. However, in some models (Qwen3, Llama3.2, and Olmo2), token generation throughput declined by 30-66% compared to the unaccelerated build, indicating a performance trade-off despite the overall acceleration gains. For example, Llama3.2, which had the lowest inference speed with 20.8 t/s prompt processing and 12.18 t/s token generation, achieved a notable speedup after deploying to OpenCL. Prompt processing speed increased to 110 t/s, representing a 5x acceleration. However, token generation speed declined to 8.7 t/s, indicating a performance trade-off in inference

processing. In contrast, Gemma3 could gain acceleration for both token generation and prompt processing, adding 116 t/s for prompt processing and 4 t/s for token generation. The largest acceleration was obtained by Olmo2, resulting in 8.5x quicker prompt processing (127 t/s) with a 25% token generation speed trade-off (8.28 t/s). Another trade-off was observed in model load time. By offloading the models to the GPU, they took 2–3 seconds longer to load.

Table 8. GLEU Scores for Different Models on Server and Mobile Devices

Device	Type	Qwen3	Gemma3	Llama3.2	Olmo2
A40	Server	0.8056	0.8048	0.8209	0.8347
Snapdragon 8 Gen 1 (4-bit + Accel.)	Mobile	0.8028	0.8115	0.8198	**0.8339**
Snapdragon 855 (4-bit)	Mobile	0.8010	0.8181	0.8213	**0.8330**

Table 9. F0.5 Scores for Different Models on Server and Mobile Devices

Device	Type	Qwen3	Gemma3	Llama3.2	Olmo2
A40	Server	0.4711	0.4473	0.5150	0.5182
Snapdragon 8 Gen 1 (4-bit + Accel.)	Mobile	0.4385	0.4314	0.4762	**0.5182**
Snapdragon 855 (4-bit)	Mobile	0.4270	0.4314	0.4762	**0.5150**

Table 10. LLM Scores for Different Models on Server and Mobile Devices

Device	Type	Qwen3	Gemma3	Llama3.2	Olmo2
A40	Server	7.7	7.9	7.2	7
Snapdragon 8 Gen 1 (4-bit + Accel.)	Mobile	7.5	7.5	7	7
Snapdragon 855 (4-bit)	Mobile	7.5	7.7	7	7

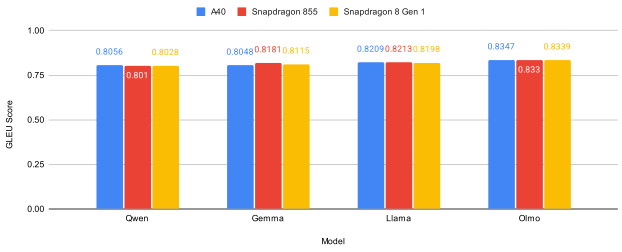


Figure 15. GLEU Score Comparison Server-Mobile

Employing hardware acceleration influenced model accuracy performance on mobile devices. Across all models and both devices, the OpenCL delegate affected GLEU and F0.5 by less than 1% in most cases and LLM evaluation by less than 0.5 points, indicating that acceleration has little to no practical impact on correction quality. Differences in GLEU scores between mobile devices and the server are summarized in Table 8, while variations in F0.5 scores are shown in Table 9. Additionally, Table 10 presents the LLM-based evaluation score differences between mobile and server predictions.

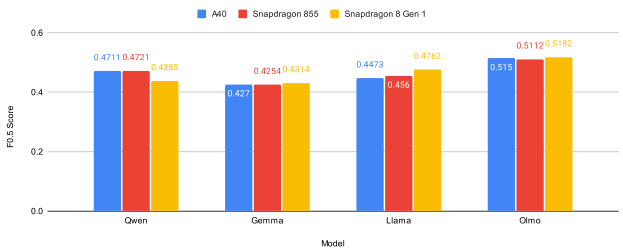


Figure 16. F0.5 Score Comparison Server-Mobile

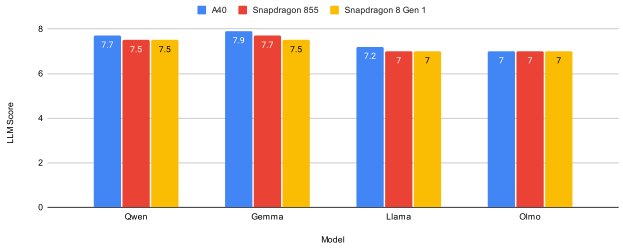


Figure 17. LLM Score Comparison Server-Mobile

For the Snapdragon 8 Gen 1 (Figure 15), GLEU scores showed minor accuracy losses, less than 0.4% for Qwen3, Llama3.2, and Olmo2 models. In contrast, Gemma3 experienced an accuracy improvement of approximately 0.83%. Regarding F0.5 scores (Figure 16), only Qwen3 showed a notable accuracy decrease (around 6.9%), whereas the remaining models demonstrated accuracy gains, with Llama3.2 achieving the highest increase of approximately 6.46%.

A different pattern emerged for the Snapdragon 855 device. GLEU scores indicated slight accuracy decreases of less than 0.6% for Qwen3 and Olmo2. Conversely, Gemma3 gained roughly 1.6%, and Llama3.2 marginally improved by about 0.05%. Llama3.2 and Qwen3 showed improvements in F0.5 scores, with gains of approximately 2% and 1.1%, respectively. In contrast, Olmo2 and Gemma3 experienced slight declines of around 0.7% and 0.4%.

For LLM-based evaluation (Figure 17), all models exhibited minor degradation, with score differences of less than 0.4 points when compared to server outputs, with Olmo2 showing no change on both mobile devices. Both Qwen3 and Llama3.2 obtained a 0.2 point reduction, while Gemma3 had its LLM evaluation score lowered by 0.4 and 0.2 points for Snapdragon 8 Gen 1 and Snapdragon 855, respectively.

4.4 Application Modules Latency

Three key components of the AR grammar-checker pipeline (OCR, LLM, AR) were profiled on two devices: a Pixel 4 (Snapdragon 855) running 4-bit models without hardware acceleration and a OnePlus 10 Pro (Snapdragon 8 Gen 1)

Table 11. Module Latency (ms) on Snapdragon 8 Gen 1 and Snapdragon 855

Device	Module	Qwen3	Gemma3	Llama3.2	Olmo2
Snapdragon 8 Gen 1 (4-bit + Accel.)	OCR	188.80	140.00	133.60	130.40
	TTFT	401.80	498.40	494.80	380.20
	AR	5.20	4.60	4.40	4.20
	Total	595.80	643.00	632.80	**514.80**
Snapdragon 855 (4-bit)	OCR	67.00	44.40	48.20	58.40
	TTFT	2402.60	2461.00	4300.40	2812.80
	AR	1.40	1.20	1.20	0.60
	Total	**2471.00**	2506.60	4349.80	2871.80

using the same 4-bit models with the on-device accelerator enabled. The latency results are summarized in Table 11.

For the LLM module, latency is reported as Time to First Token (TTFT), measured by inputting a 15-word sentence to the model after OCR extraction. OCR latency differed modestly, averaging 148.2 ms on the Snapdragon 8 Gen 1 and 54.5 ms on the Snapdragon 855. AR rendering was negligible on both platforms, remaining below 5 ms for all devices.

The LLM stage dominated end-to-end latency. On the Snapdragon 8 Gen 1, TTFT ranged from 380.2 ms (Olmo2) to 498.2 ms (Gemma3). On the Snapdragon 855, the corresponding range was substantially higher, 2471 ms (Qwen3) to 4349.8 ms (Llama3.2). Aggregating all modules, the Snapdragon 8 Gen 1 Pro achieved its fastest total latency with Olmo2 (514.8 ms), whereas on the Snapdragon 855, the lowest total was obtained with Qwen3 (2471.8 ms).

Table 12. MediaPipe inference comparison on Snapdragon 8 Gen 1 and Snapdragon 855

Device	Engine	GLEU	F0.5	TTFT (ms)	tg (t/s)
Snapdragon 8 Gen 1	CPU	0.5053	0.0021	2884.16	26.7954
	GPU	0.5655	0.0057	1156.46	8.7424
Snapdragon 855	CPU	0.5102	0.0089	5792.04	15.5026
	GPU	0.5709	0.0328	3067.35	16.8959

Under the MediaPipe framework, Table 12, only a Gemma3 4-bit quantized model was available for evaluation, as fine-tuned LoRA-based quantized models failed to produce output. All evaluations were conducted directly on-device using the full test set because no server-side testing was supported. On the Snapdragon 8 Gen 1, token generation speed averaged 26.8 t/s on the CPU and 8.7 t/s on the GPU, despite both configurations exhibiting comparable TTFT performance. In contrast, the gap in token generation between the GPU and CPU models was not significant in Snapdragon 855, with the GPU model being around 1.5 t/s faster. Since the MediaPipe model employed a few-shot setup, its average GLEU score closely aligned with the Gemma3 model evaluated via llama.cpp, averaging around 0.5. Nonetheless, the F0.5 score was significantly low in every case, remaining below 0.033.

These results underline that hardware acceleration delivers a decisive improvement for 4-bit LLM inference on mobile devices. However, even with quantization, the LLM stage remains the primary latency bottleneck.

4.5 Usability Testing Results

User evaluation was conducted with 18 Indonesian high school students in a classroom setting (Appendix A.6) and resulted in marginally acceptable usability. Figure 18 illustrates the SUS score distribution among students. Table 14 shows the average SUS score and per-item point. The user evaluation achieved a SUS mean of 59.3 ± 15.1, placing the application in the "OK" / "marginally acceptable" band [6]. Five students graded the application usability below "OK", with one student rating it as "Awful" (<35.7 SUS score). On the other hand, only three students felt the application had "Good" usability. The majority of students gave SUS scores between the "OK" and "Good" bands.

Table 13. Average SUS Point and Score

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	**Score**
Avg	3.78	3.00	3.56	3.28	4.06	2.44	3.83	2.50	3.33	3.61	59.31

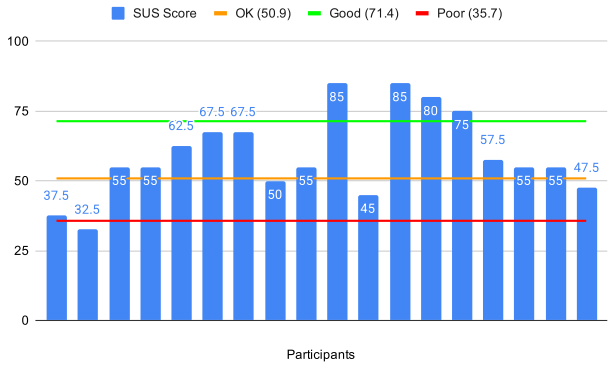


Figure 18. SUS Score Distribution

Based on class observation and student feedback, some students needed guidance and time to learn how to use the application. This finding aligns with the average scores of 3.28 for SUS Q4 and 3.61 for SUS Q10, respectively. Moreover, there was a complaint about the application being slow and heavy. The use of a non-accelerated build for testing, with a latency of approximately 2.5 seconds, could potentially influence this issue. However, students liked the grammar checker feature, which could help them correct their writing. In addition, they suggested iOS support in the future.

5 Discussion

5.1 Answers to Research Questions

This study demonstrates quantization and hardware acceleration can optimize time behavior in AR grammar checkers

while preserving accuracy. In general, the results indicated that quantization improved inference speed with an accuracy drop. Hardware acceleration significantly enhanced prompt processing, but it came with a trade-off in model load time and token generation that varied depending on the models. Moreover, the combination of quantization and hardware accelerator could speed up inference while maintaining model accuracy.

To address the research questions, the following findings are presented:

5.1.1 RQ1: What strategies can be implemented to improve the time behavior performance of an AR grammar checker application while maintaining accuracy?

Based on the findings, there are two configuration options that suit the AR grammar checker, depending on specific requirements. Users can consider these configurations based on their device specifications.

First, for a device without acceleration, choosing the 4-bit model is the best option to strike a balance between accuracy, speed, and memory. The recommended model is Olmo2, with a less sensitive trade-off between accuracy and speed.

The second option is for a device that supported hardware acceleration and considered latency more critical than accuracy. The Olmo2 model, when using 4-bit precision, could achieve faster inference speeds but would experience a slightly different drop in accuracy compared to the 8-bit model. This solution meets the 300–500 ms latency window typically required for regular tapping tasks [36], with the LLM module itself accounting for 380.2 ms.

5.1.2 RQ1.1: What is the effect of different quantization bits on the time behavior performance and accuracy of the LLM in the AR grammar checker?

The results highlight that applying quantization can improve time behavior at the cost of some accuracy. 8-bit PTQ improved prompt processing and token generation, while the model load time results varied depending on the model. Regarding accuracy, 8-bit PTQ insignificantly reduced all metrics by less than 1.5%, suggesting that the models were able to maintain their accuracy.

In 4-bit quantization, prompt processing was slightly lower than 8-bit for all models and devices. The Snapdragon 855, on the other hand, improved token generation by about 5–20 percentage points. On Snapdragon 8 Gen 1, token generation was slower for most models. Additionally, some models load faster and others load slower. Model accuracy dropped more in 4-bit PTQ relative to f16, <0.15% for GLEU, <4% for F.0.5, and <5% LLM evaluation score for most models. Gemma3 was sensitive to change, noting 3.18% GLEU drops and 18% F0.5 drops.

5.1.3 RQ1.2: In what ways does hardware acceleration further enhance the time behavior performance of the AR grammar checker?

Hardware acceleration significantly increased the speed of prompt processing, but it came at the cost of slower token generation and model loading speeds. Prompt processing could be improved by 6–20x compared to base f16 models. In contrast, token generation was reduced by 30-66% compared to 4-bit models without an accelerator. Moreover, offloading the model to the GPU required more time, as it could add 2–3 seconds of model loading time.

5.1.4 RQ1.3: How do combinations of quantization and hardware accelerators affect the overall latency and accuracy of the AR grammar checker?

Hardware acceleration is limited to 4-bit models only. There was no major accuracy drop by employing a hardware accelerator for all metrics (below 1%), pointing to maintainable correction accuracy. Furthermore, the combination of a 4-bit PTQ and OpenCL accelerator could achieve less than 500 ms of latency for all models, with Olmo2 emerging as the most robust model across scenarios due to stable performance trade-offs.

5.1.5 RQ2: How usable is an AR-based grammar checker for students in Indonesia?

A user testing in a classroom setting produced a mean System Usability Scale of 59.3 with a standard deviation of 15.1. According to Bangor's adjective ratings [6], a score between 51 and 68 is considered "OK" / "marginally acceptable". Qualitative comments included perceptible lag, a high learning curve, assistance dependence, and adding iOS device supports.

5.2 OCR Robustness

Despite fine-tuning Tesseract OCR for enhanced accuracy, executing offline OCR on-device continues to pose several challenges. The quality of OCR output is very dependent on how the user takes the picture and the environmental conditions. Skewed pictures, low-light pictures, or blurred pictures are some of the conditions that could affect OCR accuracy. These problems may require users to recapture the image or recheck OCR extraction.

Careful preprocessing can be carried out to improve the quality of the input. One of the widely used libraries for preprocessing is OpenCV, and integrating OpenCV into the application can add around 100 MB in size, while the OCR model itself (Tesseract OCR/MLKit) occupies just about 15 MB. Furthermore, by leveraging vision LLMs such as TrOCR, it could also be explored to gain more robust OCR results. However, compatibility for running vision LLMs on-device

has to be considered due to the resource-constrained environment and inference library support. Adding more computation and memory may not be suitable for the users who have limited-resource devices.

Another challenge is people's handwriting varies, and covering different styles requires more samples for the training data. In contrast, the availability of public annotated handwriting is limited, and the process of creating the dataset from scratch will require more resources.

5.3 Quantization Trade-offs

In general, quantization consistently improved inference speed, but its impact on accuracy varied. The GLEU score had degraded less than 3.2% across all models, while the F0.5 score was more sensitive to precision change, up to 18.2% for the Gemma3 model. The Llama3.2 was less impacted after quantization with a minor F0.5 score drop (<1.38%) and even led to an increase in accuracy for GLEU scores. This behavior aligns with findings from previous studies, which suggest that quantization may enhance generalization by introducing noise into model weights, thereby reducing overfitting [4, 22]. A different result was found in the LLM evaluation score, where Qwen3, Gemma3, and Llama3.2 improved their scores after 8-bit quantization, while Olmo2 showed the opposite. A possible cause of this behavior is that Llama3-70B does not necessarily score grammar correction based on the input but also takes overall fluency into account. In 4-bit quantization, all models had an accuracy decrease of less than 5%.

Inference time varied depending on the model, but in general, quantization improved the overall speed of the model when deployed to a mobile device. Even though 4-bit quantization reduced precision and model size, the results indicated that 8-bit was superior in speed to 4-bit quantization. The highest speed gain was Olmo2, with 167.78% speedup on its 8-bit quantized model and 144% on 4-bit precision. I assumed that it was either the framework (llama.cpp) or mobile CPUs that were more optimized for running 8-bit precision because its performance closely matched the f16 precision model.

The practical implication of these findings is that learners can gain fast grammar feedback, which is critical for user experience, but risk receiving incorrect corrections with some models. Gemma3's high GLEU and F0.5 degradation could disturb learning by suggesting grammatical misconceptions, while Llama3.2/Olmo2 provide balanced speed and reliability. However, regardless of the models, learners should validate the results of the grammar checker with their teacher to minimize mistakes and optimize learning outcomes.

5.4 Hardware Accelerator Impact

As discussed in the previous section, 8-bit quantization was the optimum model for speed and accuracy balance for CPU deployment. However, hardware acceleration added more insight when latency was more critical. OpenCL off-loading increased prompt-processing throughput of the 4-bit model by up to 20x relative to the f16 baseline. Although the prompt processing was improved, some models experienced a speed degradation in token generation that ranged from 30% to 66% compared to the unaccelerated model. Qwen3 experienced the worst reduction in token generation, going from 20 t/s to 6.7 t/s after acceleration. Moreover, offloading the model to the GPU took more time than loading the model to the CPU. The full loading of the model took approximately 2-3 more seconds. However, this initial delay could be a favorable trade-off for users, as it occurs only once at startup, while subsequent interactions, particularly prompt processing, benefit from significantly faster grammar feedback. The limitation of this technique was its support for only a 4-bit model and the limited range of compatible devices. As per this research that was carried out, the official documentation stated that this acceleration was only verified on high-end devices (Snapdragon 8 Gen 3 and Snapdragon 8 Elite). This research offered further validation for devices with lower specifications, as it demonstrated successful operation on the Snapdragon 8 Gen 1. I also tested the inference on Snapdragon 855, but it was unable to run due to unsupported acceleration. These compatibility issues may reduce accessibility scope for the learners because some of them may not benefit from faster responses. However, the application continues to function on the device even without an accelerator.

5.5 Usability Insights

A user study resulted in an SUS score of 59.3 ± 15.1 ("OK" / "marginally acceptable"). Although the score confirms the basic viability of the concept, it also signals that the current prototype falls short of the "Good" usability threshold of 71.4. Item-level SUS analysis and qualitative feedback highlighted several problems during the testing. Users seemed to require time to comprehend the technical aspects of the application. This issue was caused by unfamiliarity with the AR application and lack of tutorial information inside the prototype. Adding an introductory tutorial before the actual features might help ease users' learning of how to use the application. Some users were also frustrated because the application felt heavy and slow. The application used in the user study was accelerator-disabled, resulting in slower feedback in the grammar correction (around 2.5-second latency). This finding suggests that the AR grammar checker requires more optimization for faster feedback, especially for users who have low-specification devices that make them unable to use the accelerator. Additionally, this application is unsupported on iOS and requires a minimum of Android 13. Adding more device support could be considered to provide more access to students.

5.6 Limitations

While this research contributes valuable insights for integrating LLM grammar checkers in AR environments, I acknowledge some limitations in this study. Firstly, a limited

number of Indonesian error samples were used to fine-tune the LLMs, which might not fully represent common Indonesian mistakes. Moreover, a lack of human expert judgment in grammar accuracy might provide some biases to overall fluency or different possible corrections. Furthermore, the framework and testing device limit could not cover broader applications in general. This study also could not cover all sub-1B LLMs available that might have different results. Additionally, some unexpected computational overhead might exist during the testing in the application that could influence the performance measurements. Moreover, OCR engines were limited to MLKit and Tesseract OCR, which had limited handwriting datasets for fine-tuning. Finally, the user study only involved a small sample of Indonesian students in a classroom setting.

6 Conclusion

In conclusion, this study demonstrated a comprehensive approach to integrating LLMs into an offline AR grammar checker application on resource-constrained devices. The research explored quantization and hardware acceleration, resulting in significant improvements in time behavior with minimal trade-offs in grammatical correction accuracy.

8-bit quantization could enhance time behavior while maintaining model accuracy, with less than 1.5% drops. However, 4-bit quantization led to a greater reduction in accuracy, particularly for the Gemma3 model. Hardware acceleration substantially boosted prompt processing throughput by up to 20x compared to the f16 baseline with slower token generation and model loading time. Token generation degraded by 30-66%, and model load time slowed 2-3 seconds, compared to the unaccelerated model. Nevertheless, accelerated models could achieve low-latency performance within 300-500 ms with <1% degradation in grammar accuracy, achieving practical latency levels required for seamless AR experiences for maintaining learners' attention. Olmo2 emerged as an optimum configuration for both general devices and hardware-accelerated devices, balancing time behavior and accuracy effectively. In addition, the user evaluation produced a mean SUS score of 59.3 ± 15.1, categorized as "OK" / "marginally acceptable". Qualitative feedback suggested application improvements in reducing latency, adding tutorials, and broadening device compatibility.

Further research should explore more robust, tailored datasets to address a broader range of common mistakes made by Indonesian learners. Moreover, performing larger tests on various device specifications could ensure the reliability of the system. Adding more frameworks and LLMs to be tested could be done in the future to gain deeper insight about the potential configurations for the AR grammar checker. In addition, I also suggest exploring more optimization techniques that can be implemented to gain a more efficient LLM while preserving its accuracy in offline conditions or even real-time feedback. Another study can be carried out focusing on the offline OCR, especially in handwritten text, to develop a more robust grammar checker application.

7 Acknowledgments

First of all, Alhamdulillah, all praises are to Allah SWT, who has given me grace and strength to complete this research. I would like to express my sincere gratitude to my supervisors, Nacir Bouali and Faizan Ahmed, for invaluable guidance and support throughout this research. They gave me insightful feedback and encouragement to complete this research. I would also like to thank the Ministry of Communications and Informatics of the Republic of Indonesia for giving me a life-changing experience by providing me with a scholarship for my master's degree at the University of Twente. Lastly, I extend my heartfelt gratitude to my family in Indonesia and my second family here for their unwavering support throughout this journey.

References

[1] Maulana Akbar and Gustaf Wijaya. Digital literacy of rural areas in indonesia: Challenges and opportunities. In Proceedings of the 4th International Conference on Rural Socio-Economic Transformation, RUSET 2023, 1 November 2023, Bogor, Indonesia, 2024.

[2] Yousif Alshumaimeri and Noman Mazher. Augmented reality in teaching and learning english as a foreign language: A systematic review and meta-analysis. World Journal of Advanced Research and Reviews, 19(1):1093–1098, 2023.

[3] Zainab Asimiyu. Evaluating android architectural patterns: A deep dive into mvp, mvvm, and mvi. 2024.

[4] MohammadHossein AskariHemmat, Reyhane Askari Hemmat, Alex Hoffman, Ivan Lazarevich, Ehsan Saboori, Olivier Mastropietro, Sudhakar Sah, Yvon Savaria, and Jean-Pierre David. Qreg: On regularization effects of quantization. arXiv preprint arXiv:2206.12372, 2022.

[5] Dilobarkhon Azimova and Dilyorjon Solidjonov. Learning english language as a second language with augmented reality. Qo 'Qon Universiteti Xabarnomasi, 1:112–115, 2023.

[6] Aaron Bangor, Philip Kortum, and James Miller. Determining what individual sus scores mean: Adding an adjective rating scale. Journal of usability studies, 4(3):114–123, 2009.

[7] Neil A Bradbury. Attention span during lectures: 8 seconds, 10 minutes, or more? Advances in physiology education, 40 4:509–513, 2016. URL https://api.semanticscholar.org/CorpusID:25222161.

[8] J Brooke. Sus: A quick and dirty usability scale. Usability Evaluation in Industry, 1996.

[9] Christopher Bryant, Mariano Felice, and Ted Briscoe. Automatic annotation and evaluation of error types for grammatical error correction. In Regina Barzilay and Min-Yen Kan, editors, Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 793–805, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1074. URL https://aclanthology.org/P17-1074/.

[10] Christopher Bryant, Zheng Yuan, Muhammad Reza Qorib, Hannan Cao, Hwee Tou Ng, and Ted Briscoe. Grammatical error correction: A survey of the state of the art. Computational Linguistics, 49(3): 643–701, 2023.

[11] Victor R Basili-Gianluigi Caldiera and H Dieter Rombach. Goal question metric paradigm. Encyclopedia of software engineering, 1(528-532):6, 1994.

[12] I-Chun Chen. The application of augmented reality in english phonics learning performance of esl young learners. In 2018 1st International Cognitive Cities Conference (IC3), pages 255–259, 2018. doi: 10.1109/IC3.2018.000-7.

[13] Daniel Dahlmeier, Hwee Tou Ng, and Siew Mei Wu. Building a large annotated corpus of learner english: The nus corpus of learner english. In Proceedings of the eighth workshop on innovative use of NLP for building educational applications, pages 22–31, 2013.

[14] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. arXiv e-prints, pages arXiv–2407, 2024.

[15] Tao Fang, Tianyu Zhang, Derek F Wong, Keyan Jin, Lusheng Zhang, Qiang Zhang, Tianjiao Li, Jinlong Hou, and Lidia S Chao. Llmcl-gec: Advancing grammatical error correction with llm-driven curriculum learning. Expert Systems with Applications, 279:127397, 2025.

[16] Varunyu Fuvattanasilp, Matjaž Kljun, Hirokazu Kato, and Klen Čopič Pucihar. The effect of latency on high precision micro instructions in mobile ar. In 22nd International Conference on Human-Computer Interaction with Mobile Devices and Services, MobileHCI '20, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380522. doi: 10.1145/3406324.3410716. URL https://doi.org/10.1145/3406324.3410716.

[17] H. Jahangir, S. K. Goel, and S. Khurana. Scaling Up the Transformers: A Survey of Training and Inference Optimization Techniques. In 2024 International Conference on Electrical Electronics and Computing Technologies (ICEECT), volume 1, pages 1–6, August 2024. doi: 10.1109/ICEECT61758.2024.10739061. Journal Abbreviation: 2024 International Conference on Electrical Electronics and Computing Technologies (ICEECT).

[18] Torin Hopkins, Suibi Che-Chuan Weng, Rishi Vanukuru, Emma Wenzel, Amy Banic, Mark D Gross, and Ellen Yi-Luen Do. Studying the effects of network latency on audio-visual perception during an ar musical task. In 2022 IEEE International Symposium on Mixed and Augmented Reality (ISMAR), pages 26–34, 2022. doi: 10.1109/ISMAR55827.2022.00016.

[19] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

[20] Xinyi Huang, Di Zou, Gary Cheng, and Haoran Xie. A systematic review of ar and vr enhanced language learning. Sustainability, 13:4639, 04 2021. doi: 10.3390/su13094639.

[21] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2704–2713, 2018.

[22] Saqib Javed, Hieu Le, and Mathieu Salzmann. Qt-dog: Quantization-aware training for domain generalization. arXiv preprint arXiv:2410.06020, 2024.

[23] Yongkweon Jeon, Chungman Lee, Kyungphil Park, and Ho-young Kim. Z-FOLD: A frustratingly easy post-training quantization scheme for LLMs.

[24] Masamune Kobayashi, Masato Mita, and Mamoru Komachi. Large language models are state-of-the-art evaluator for grammatical error correction, 2024. URL https://arxiv.org/abs/2403.17540.

[25] Minghao Li, Tengchao Lv, Jingye Chen, Lei Cui, Yijuan Lu, Dinei Florencio, Cha Zhang, Zhoujun Li, and Furu Wei. Trocr: Transformer-based optical character recognition with pre-trained models, 2022. URL https://arxiv.org/abs/2109.10282.

[26] Enrui Liu, Changhao Liu, Yang Yang, Shanshan Guo, and Su Cai. Design and implementation of an augmented reality application with an english learning lesson. In 2018 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE), pages 494–499, 2018. doi: 10.1109/TALE.2018.8615384.

[27] Victor Marrahí-Gómez and Jose Belda-Medina. The effect of using ar technology on language learning. International Journal of English Language Studies, 2023. URL https://api.semanticscholar.org/CorpusID:256837728.

[28] U-V Marti and Horst Bunke. The iam-database: an english sentence database for offline handwriting recognition. International journal on document analysis and recognition, 5:39–46, 2002.

[29] Manan Modi, Arpita Shah, Deep Kothadiya, and Mrugendra Rahevar. Optical character recognition: Comparative analysis of tesseract and textract on diverse datasets. In 2024 3rd International Conference on Automation, Computing and Renewable Systems (ICACRS), pages 913–919, 2024. doi: 10.1109/ICACRS62842.2024.10841500.

[30] Jakub Náplava and Milan Straka. Grammatical error correction in low-resource scenarios. In Wei Xu, Alan Ritter, Tim Baldwin, and Afshin Rahimi, editors, Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019), pages 346–356, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-5545. URL https://aclanthology.org/D19-5545/.

[31] Courtney Napoles, Keisuke Sakaguchi, Matt Post, and Joel Tetreault. GLEU without tuning. eprint arXiv:1605.02592 [cs.CL], 2016. URL http://arxiv.org/abs/1605.02592.

[32] Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, et al. 2 olmo 2 furious. arXiv preprint arXiv:2501.00656, 2024.

[33] I. Panopoulos, S. Nikolaidis, S.I. Venieris, and I.S. Venieris. Exploring the Performance and Efficiency of Transformer Models for NLP on Mobile Devices. volume 2023-July, 2023. doi: 10.1109/ISCC58397.2023.10217850. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85172017980&doi=10.1109%2fISCC58397.2023.10217850&partnerID=40&md5=07ed543dbb49ba3350fd00513fb1a1f1.

[34] Puspita. Error analysis of indonesian grammatical interference in students' english composition. Buletin Poltanesa, 22, 06 2021. doi: 10.51967/tanesa.v22i1.466.

[35] Vipul Raheja, Dhruv Kumar, Ryan Koo, and Dongyeop Kang. Coedit: Text editing by task-specific instruction tuning. 2023.

[36] Walter Ritter, Guido Kempter, and Tobias Werner. User-acceptance of latency in touch interactions. pages 139–147, 08 2015. ISBN 978-3-319-20680-6. doi: 10.1007/978-3-319-20681-3_13.

[37] StatCounter. Mobile operating system market share indonesia. URL https://gs.statcounter.com/os-market-share/mobile/indonesia.

[38] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices. arXiv preprint arXiv:2004.02984, 2020.

[39] Gemma Team. Gemma 3. 2025. URL https://goo.gle/Gemma3Report.

[40] Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[41] Yash Thakare, Tejas Sridhar, Navanit Srisangkar, and Pankaj Vanwari. Application for grammar checking and correction. 2020.

[42] Oguzhan Topsakal and Elif Topsakal. Framework for a foreign language teaching software for children utilizing ar, voicebots and chatgpt (large language models). The Journal of Cognitive Systems, 7(2):33–38, 2022.

[43] Satoru Uchida, Yuki Arase, and Tomoyuki Kajiwara. Profiling english sentences based on cefr levels. ITL-International Journal of Applied Linguistics, 175(1):103–126, 2024.

[44] Alla Durga Umesh Chandra, Govindolla Sricharan, Challa Satya Sai Durga Kalyan, and Badri Venkata Sai Kiran. Design and development of offline handwritten text recognition leveraging machine learning techniques. In 2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS), pages 1328–1333, 2022. doi: 10.1109/ICICCS53718.2022.9787986.

[45] Mei-Hung Wu. The applications and effects of learning english through augmented reality: A case study of pokémon go. Computer Assisted Language Learning, 34(5-6):778–812, 2021.
[46] Helen Yannakoudakis, Ted Briscoe, and Ben Medlock. A new dataset and method for automatically grading esol texts. In Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies, pages 180–189, 2011.
[47] Helen Yannakoudakis, Øistein E Andersen, Ardeshir Geranpayeh, Ted Briscoe, and Diane Nicholls. Developing an automated writing placement system for esl learners. Applied Measurement in Education, 31(3):251–267, 2018.
[48] Xun Yuan, Derek Pham, Sam Davidson, and Zhou Yu. ErAConD: Error annotated conversational dialog dataset for grammatical error correction. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz, editors, Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 76–84, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.5. URL https://aclanthology.org/2022.naacl-main.5/.
[49] Yunisrina Qismullah YUSUF, Faisal MUSTAFA, and Rizky Muhammad IQBAL. An inquiry into grammatical errors in writing committed by high achieving efl students. International journal of language studies, 15(2), 2021.
[50] Min Zeng, Jiexin Kuang, Mengyang Qiu, Jayoung Song, and Jungyeul Park. Evaluating prompting strategies for grammatical error correction based on language proficiency, 2024. URL https://arxiv.org/abs/2402.15930.

A Appendix

A.1 Few-shot Prompt Template

Fix grammatical error, only output corrected sentece.
Examples:

1. Fix grammar: Sometimes the employers do strike and we clients do not known in time . As a result , we can not get on time to work . That is one of the disadvantages of using public transport .
Sometimes the staff go on strike and we customers do not know in time . As a result , we can not get to work on time . That is one of the disadvantages of using public transport .

2. Fix grammar: I definitely agree with that .
I definitely agree with that .

3. Fix grammar: They should have healthy habits and , with their examples , they can improve the habits of the population .
They should have healthy habits and , by their example , they can improve the habits of the population .

4. Fix grammar: Suddenly , he heard something . Was it an animal ? , he thinked .
Suddenly , he heard something . Was it an animal ? , he thought .

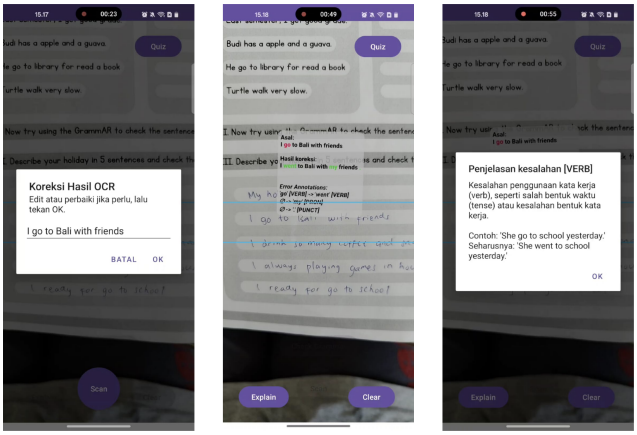
5. Fix grammar: Continue the study
Continue studying

Now correct this:
6. Fix grammar: <input_sentence>

A.2 Llama3-70B Evaluation Prompt Template

You're a grammar expert. Here are N pairs of incorrect and corrected sentences. Please rate the overall grammatical correction quality of this batch from 1 (very poor) to 10 (perfect). Only return a single number, no explanation.
Example 1: Incorrect: <src_1> Corrected: <pred_1>
...
Example N: Incorrect: <src_N> Corrected: <pred_N>
Score:

A.3 Application User Interfaces



A.4 Explanation templates for annotated error types in Bahasa

"SYM": "Kesalahan penggunaan simbol, seperti tanda mata uang, persen, atau lainnya yang tidak tepat. Contoh: 'He paid $20dollars.' Seharusnya: 'He paid $20.'",
"PUNCT": "Kesalahan dalam penggunaan tanda baca seperti kurangnnya titik, koma, tanda tanya, dan lainnya. Contoh: 'She is beautiful' Seharusnya: 'She is beautiful.'",
"ADJ": "Kesalahan penggunaan kata sifat (adjective), seperti salah bentuk perbandingan atau penggunaan kata sifat pada posisi yang tidak tepat. Contoh: 'This is more better.' Seharusnya: 'This is better.'",
"CCONJ": "Kesalahan penggunaan kata hubung koordinatif (coordinating conjunction) seperti 'and', 'but', 'or'. Contoh: 'I like tea, but also I like coffee.' Seharusnya: 'I like tea, and I also like coffee.'",
"NUM": "Kesalahan penggunaan angka atau bentuk bilangan (number), termasuk bentuk tunggal/jamak atau pemformatan. Contoh: 'She have two childrens.' Seharusnya: 'She has two children.'",

"DET": "Kesalahan pemilihan atau penggunaan kata sandang (determiner) seperti a, an, the, atau kata penentu lainnya. Contoh: 'I saw cat on street.' Seharusnya: 'I saw a cat on the street.'",
"PRON": "Kesalahan penggunaan kata ganti (pronoun) seperti he, she, it, they, dan sebagainya. Contoh: 'My brother lost her book.' Seharusnya: 'My brother lost his book.'",
"X": "Kesalahan yang tidak dapat dikategorikan dalam kelas kata tertentu, biasanya mencakup kata tidak dikenal atau penggunaan yang sangat tidak wajar. Contoh: 'She blargh the file.' — kata 'blargh' tidak dikenali.",
"ADP": "Kesalahan dalam penggunaan kata depan (preposition) seperti in, on, at, to, dan lain-lain. Contoh: 'She is good in math.' Seharusnya: 'She is good at math.'",
"VERB": "Kesalahan penggunaan kata kerja (verb), seperti salah bentuk waktu (tense) atau kesalahan bentuk kata kerja. Contoh: 'She go to school yesterday.' Seharusnya: 'She went to school yesterday.'",
"NOUN": "Kesalahan penggunaan kata benda (noun), seperti bentuk tunggal dan jamak. Contoh: 'She has three cat.' Seharusnya: 'She has three cats.'",
"PROPN": "Kesalahan penulisan huruf kapital pada kata nama diri (proper noun). Contoh: 'i visited paris.' Seharusnya: 'I visited Paris.'",
"ADV": "Kesalahan penggunaan kata keterangan (adverb), seperti penempatan yang salah atau penggunaan bentuk kata yang tidak sesuai. Contoh: 'He runs quick.' Seharusnya: 'He runs quickly.'",
"INTJ": "Kesalahan penggunaan kata seru (interjection) seperti 'oh', 'wow', 'ouch', terutama dalam konteks atau penulisan. Contoh: 'Wow! you did it?' Seharusnya: 'Wow! You did it!'",
"AUX": "Kesalahan pada kata kerja bantu (auxiliary verb) seperti 'is', 'are', 'was', 'were', 'has', 'have'. Contoh: 'He have eaten.' Seharusnya: 'He has eaten.'",
"SCONJ": "Kesalahan penggunaan kata hubung subordinatif (subordinating conjunction) seperti 'because', 'although', 'when'. Contoh: 'I stayed home although it was raining.' Jika struktur kalimat kacau, itu bisa dikategorikan di sini.",
"PART": "Kesalahan penggunaan partikel dalam frasa kata kerja (phrasal verb). Contoh: 'Please look the word in dictionary.' Seharusnya: 'Please look up the word in the dictionary.'"

A.5 User Testing Worksheet

LEARNING GRAMMAR

I. Find the 'grammar error' words.

- He borrow pencil from she.
- I am study english in school
- Last semester, I get good grade.
- Budi has a apple and a guava.
- He go to library for read a book
- Turtle walk very slow.

II. Now try using the GrammAR to check the sentence.

III. Describe your holiday in 5 sentences and check the grammar.

A.6 SUS Questionnaire Responses

Table 14. SUS Questionnaire Responses and Scores

Student	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Score
1	3	5	1	4	2	3	1	2	3	1	37.5
2	1	5	1	4	4	4	3	3	4	4	32.5
3	4	3	4	2	4	2	3	4	2	4	55.0
4	4	3	4	5	5	3	4	2	3	5	55.0
5	4	3	3	2	5	3	4	2	3	4	62.5
6	4	3	4	2	4	1	4	3	3	3	67.5
7	4	3	4	3	4	1	3	2	4	3	67.5
8	4	3	3	4	3	3	4	3	3	4	50.0
9	4	3	4	4	4	2	4	4	3	4	55.0
10	4	2	5	2	5	2	5	1	5	3	85.0
11	4	3	2	5	4	3	4	4	3	4	45.0
12	4	2	5	2	5	2	5	1	5	3	85.0
13	4	2	5	2	4	2	4	1	5	3	80.0
14	4	2	5	2	5	2	5	1	2	4	75.0
15	4	3	4	4	4	3	4	3	3	3	57.5
16	4	3	3	4	4	2	4	3	3	4	55.0
17	4	3	4	4	4	3	4	2	3	5	55.0
18	4	3	3	4	3	3	4	4	3	4	47.5