Deploying Language Models on Android-Based Edge Devices: A Practical Evaluation Pipeline

Suayder Costa[1], Igor Lima[1], William Harada[1], Mateus Lucena[1], Arthur Alves[1], Myke Valadão[1], Cassio Alves[1], Ruan Belem[2], Agemilson Pimentel[2], Romulo Fabricio[2], Alexandre Miranda[3], Daniel Lins[1], Frederico Gonçalves[1], Sidney Leal[1]

[1] Venturus – Innovation & Technology, Manaus, Brazil

[2]TPV Technology, Manaus, Brazil

[3]Paulo Feitoza Foundation - FPFTech, Manaus, Brazil

{suayder.costa, igor.lima, william.harada}@venturus.org.br

{myke.valadao, cassio.alves, arthur.alves}@venturus.org.br

{daniel.lins, frederico.goncalves, sidney.leal}@venturus.org.br

{ruan.belem, agemilson.pimentel, romulo.fabricio}@tpv-tech.com

alexandre.miranda@fpf.br, lucena.mateo@gmail.com

Abstract. *This paper presents a practical evaluation pipeline for deploying small language models (SLMs), compact versions of large language models designed to run with reduced memory and computational requirements, on constrained Android-based edge devices, using Android TV as a representative case. The study investigates both deployment feasibility and software-level acceleration strategies, such as the choice of the inference engine responsible for executing the model on the device, under strict memory and processing limitations. Results show that most models above 500 million parameters were not suitable for the target environment, while a subset of 4-bit quantized models achieved stable execution and acceptable response quality. In addition, the experiments demonstrate that the choice of inference engine has a major impact on performance, with MNN significantly outperforming llama.cpp on ARM-based devices (the processor family that dominates mobile and embedded hardware). These findings provide practical guidance for integrating generative AI into low-resource consumer hardware.*

Resumo. *Este artigo apresenta um pipeline prático de avaliação para a implantação de pequenos modelos de linguagem, versões compactas dos grandes modelos de linguagem projetadas para operar com menor consumo de memória e processamento, em dispositivos de borda baseados em Android, utilizando uma Android TV como caso representativo. O estudo investiga tanto a viabilidade de implantação quanto estratégias de aceleração em nível de software, como a escolha do motor de inferência responsável pela execução do modelo no dispositivo, sob severas restrições de memória e processamento. Os resultados mostram que a maioria dos modelos acima de 500 milhões de parâmetros não foi adequada ao ambiente avaliado, enquanto um subconjunto*

de modelos quantizados em 4 bits apresentou execução estável e qualidade de resposta aceitável. Além disso, os experimentos demonstram que a escolha do motor de inferência tem forte impacto no desempenho, com o MNN superando significativamente o llama.cpp em dispositivos ARM (família de processadores que predomina em hardware móvel e embarcado). Esses achados oferecem orientações práticas para a integração de IA generativa em hardware de consumo com recursos limitados.

1. Introduction

In recent years, the capabilities of large language models (LLM) have advanced significantly, and foundational large language models such as BERT [Devlin et al. 2019], GPT-3 [Brown et al. 2020], and LLaMA [Grattafiori et al. 2024] have redefined what is possible in natural language processing, expanding both language understanding and generation capabilities. These advances have strongly impacted applications such as chatbots and virtual assistants, which are now capable of handling complex user queries and maintaining context across longer interactions. As generative Artificial Intelligence (AI) becomes increasingly integrated into everyday applications, there is growing interest in moving beyond traditional cloud-based deployments and enabling these models to operate directly on user devices [Sai et al. 2024]. However, the substantial computational and memory requirements of these models still restrict their execution to high-end hardware environments.

To address this limitation, recent research has explored the development of small language models (SLM), which aim to approximate the capabilities of larger models while significantly reducing their computational footprint[1] [Allal et al. 2025]. Although SLM typically present lower overall performance compared to full-scale LLM, they are capable of producing competitive responses for many practical tasks [Hillier et al. 2024]. Deploying generative models directly on edge devices (end-user hardware itself, such as smartphones, smart TVs, or IoT devices, rather than on remote servers) offers several important advantages, including improved privacy, offline functionality, and independence from external APIs. From an industrial perspective, this approach can also reduce operational costs associated with cloud infrastructure and third-party services, making it particularly attractive for consumer electronics such as smart TVs and internet of things (IoT) systems [Bhat et al. 2024]. This scenario is particularly relevant for consumer electronics that combine high market penetration with severely limited hardware, such as Android TV devices, where adding generative capabilities locally could enrich the user experience without depending on cloud infrastructure. Despite these benefits, running even compact models on edge hardware remains challenging due to still significant resource demands.

Consequently, a large body of research has focused on reducing the computational requirements of language models through techniques such as quantization (representing model weights with fewer bits to reduce memory usage) [Zafrir et al. 2019, Dettmers et al. 2023], pruning (removing redundant parameters) [Agrawal et al. 2025], knowledge distillation (training a smaller model to imitate a larger one) [Sanh 2019, Du et al. 2025], and lightweight architectural designs [Sarah et al. 2025, Qin et al. 2024,

[1] `https://qwenlm.github.io/blog/qwen2.5/`, `https://github.com/openbmb/minicpm`

Li et al. 2024]. At the same time, specialized inference engines (software runtimes optimized to load and execute neural network models efficiently on a target platform) such as ONNX Runtime[2], TensorFlow Lite[3], MNN[4] [Jiang et al. 2020b], and llama.cpp[5] have been developed to improve execution performance across different hardware platforms. However, many existing studies primarily evaluate these approaches on relatively powerful mobile devices equipped with large amounts of RAM and dedicated computational accelerators. This differs significantly from highly constrained embedded platforms such as Android TV devices, which typically rely on low-power ARM processors, operate under 32-bit memory addressing limitations (which cap the addressable RAM at approximately 3GB and prevent the use of larger models that would otherwise fit in physical memory), and often lack specialized hardware accelerators (dedicated chips such as NPUs or GPUs designed to speed up neural network execution) [Elhanashi et al. 2024].

Furthermore, the literature still lacks systematic and comparative analyses evaluating the real-world impact of different inference engines and optimization strategies on consumer-grade embedded hardware. Open-source frameworks frequently omit detailed information regarding architectural constraints, such as memory usage patterns, CPU dependencies, or compatibility with specific embedded environments, which are critical factors in constrained platforms. Additionally, only limited empirical research has investigated the integration of language models into native Android applications, particularly within Android TV operating systems, where additional constraints arise from memory allocation policies and background execution restrictions [de Souza et al. 2025]. To address these gaps, this study proposes an evaluation pipeline (Figure 1) designed to systematically assess the deployment of SLMs on edge devices, using Android TV as a representative case of highly constrained 32-bit embedded hardware. Through this pipeline, we conduct an experimental study that provides: (i) a feasibility analysis of SLM execution on Android TV, including operational stability, memory usage, and response quality in a simulated customer support scenario; (ii) a trade-off analysis between model throughput, output quality, and response consistency to support deployment decisions in constrained environments; and (iii) a comparative evaluation of multiple inference engines and optimization strategies, highlighting implementation challenges and identifying the MNN engine as the most suitable candidate for performance on 32-bit devices.

2. Evaluation Pipeline Design

To systematically evaluate the deployment viability and performance of SLM on edge devices, this study introduces a two-phase experimental pipeline. The proposed pipeline is designed to ensure reproducibility, preserve practical relevance, and provide comprehensive insight into both model-level and system-level behaviors under resource-constrained conditions.

The pipeline is structured around three key research questions:

RQ1: Which quantized SLMs are viable for deployment on constrained edge devices such as Android TVs?

[2]https://github.com/microsoft/onnxruntime
[3]https://github.com/google-ai-edge/LiteRT
[4]https://github.com/alibaba/MNN
[5]https://github.com/ggml-org/llama.cpp

RQ2: What trade-offs exist between model size, response quality, latency, and stability in edge environments?

RQ3: To what extent can software-level optimization improve inference performance, and which components (e.g., CPU libraries or inference engines) contribute most significantly?

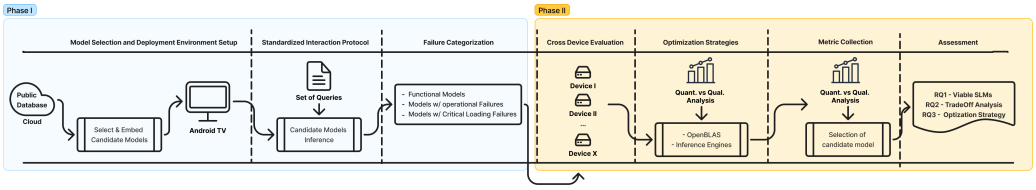

Figure 1. Evaluation pipeline for assessing SLM deployment and optimization on edge devices.

The evaluation is structured into two sequential phases, each designed to address specific research questions. Phase 1 focuses on RQ1 and RQ2, assessing the feasibility and baseline performance of various quantized SLMs on constrained edge devices. Phase 2 targets RQ3, investigating how performance is influenced by software-level acceleration strategies, such as inference engine choice and low-level library integration. Figure 1 outlines the overall structure of the proposed methodology, while each phase is discussed in detail in the following sections, as well as the details of the practical implementation for the pipeline.

2.1. Phase 1: Baseline Characterization

The first phase of the proposed pipeline is designed to identify which SLM can operate reliably under stringent memory and computational constraints. This phase establishes a standardized approach that enables reproducible evaluation across diverse hardware configurations. The following steps are applied:

1. **Model Selection:** This involves selecting representative quantized SLMs of various sizes and architectures, prioritizing instruction-following, alignment quality, and real-world edge deployment relevance. Over 20 SLMs, ranging from 65 million to 1 billion parameters, were selected for evaluation — see Table 1. The IFEval[6] scores were used to assess instruction-following capability and alignment quality, where models with high IFEval scores were prioritized. Additional candidates were selected to reflect diversity in architecture and popularity within the open-source community.
2. **Deployment Setup:** Each model is deployed within a native Android application on a constrained edge device. This controlled deployment environment ensures consistency in runtime conditions. Each model is deployed in a native Android application running on a commercial Android TV (ARM Cortex-A55, 3GB RAM), ensuring consistency in runtime conditions. All models were tested using 4-bit

[6]https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard

quantization. This decision was based on preliminary tests indicating that the 8-bit versions did not offer significant performance improvements while demanding considerably more memory, thus increasing the risk of loading failures.

3. **Interaction Protocol:** This step is important for initially filtering and ranking models for the next phase, using a standardized query set to stress the models and ensure consistent input across evaluations. Specifically, a fixed set of representative user queries (Table 2) simulates typical TV customer support use cases.
4. **Failure Categorization:** Categorizing failure modes by loadability and inference success will provide insights into deployment boundaries on resource-limited devices. Models are classified based on loadability and inference capability: (i) fully functional, (ii) fails during inference, and (iii) not loadable due to memory issues. This categorization is based on a qualitative analysis of the executions, further explained in Section 3.1.

Table 1. Selected models and their IFEval scores from the OpenLLM Leaderboard. Higher is better

Model ID	Release	Model	Params	IFEval Score
M1	2025	InternVL3-1B [Zhu et al. 2025]	1B	N/A
M2	2025	SmolLM2-360M [Allal et al. 2025]	360M	21.15%
M3	2025	MiniCPM4-0.5B [a]	0.5B	N/A
M4	2024	HyperCLOVAX-0.5B [Yoo et al. 2024]	0.5B	N/A
M5	2025	SmolLM2-135M [Allal et al. 2025]	135M	18.18%
M6	2024	Lite_Oute1-65M	65M	N/A
M7	2024	Qwen2.5-0.5B [b]	0.5B	31.53%
M8	2024	OpenELM-450M-Instruct [Mehta et al. 2024]	450M	N/A
M9	2023	Pythia410m-Instruct [Biderman et al. 2023]	410M	21.95%
M10	2024	H2o-Danube-500M [c]	500M	N/A
M11	2024	AMD-Llama-350M-Upgraded	350M	N/A
M12	2023	pythia-410m [Biderman et al. 2023]	410M	21.95%
M13	2022	flan-t5-small [Chung et al. 2024]	80M	15.24%
M14	2024	MobileLLM-600M [Liu et al. 2024]	600M	N/A
M15	2024	OpenELM-450M [Mehta et al. 2024]	450M	N/A
M16	2025	Qwen3.0-0.6B [d]	0.6B	N/A
M17	2024	InstructLM-500M [Cheng et al. 2024]	500M	10.28%
M18	2024	SmolLlamix-8x101M	808M	N/A
M19	2022	bloom-560m [Dakle et al. 2022]	560M	6.20%
M20	2024	Llama-3.2-1B-Instruct [Dubey et al. 2024]	1B	56.98%
M21	2025	Gemma-3-1B-Instruct [Team et al. 2025]	1B	N/A

[a] `https://github.com/openbmb/minicpm`
[b] `https://qwenlm.github.io/blog/qwen2.5/`
[c] `https://huggingface.co/collections/h2oai/h2o-danube3`
[d] `https://github.com/QwenLM/Qwen3`

Table 2. Representative Interaction Queries for Test Cases

ID	Question
Q1	What to do if a device connected with HDMI is not working?
Q2	How to insert the antenna plug into the 'Antenna' socket?
Q3	How do I open Netflix?
Q4	I want to talk to the Philips customer support.
Q5	Where do I find the date and time settings?

2.2. Phase 2: Software-Level Optimization

The second phase evaluates inference acceleration strategies across multiple hardware platforms:

1. **Cross-Device Benchmarking:** To ensure the consistency of model quality, a representative subset of SLMs are tested on smartphones and desktops with ARM and x86 architectures. Table 3 details all devices and their specifications used for testing. Also, Table 4 summarizes inference engine compatibility with Android architectures.
2. **Optimization Techniques:** The study evaluates optimized linear algebra libraries (e.g., OpenBLAS) and compares different inference engines, including llama.cpp,

MNN, and LiteRT, to analyze their impact on computational efficiency and compatibility across Android architectures (32-bit and 64-bit), as summarized in Table 4.

3. **Metrics:** Standardized indicators include prefill speed (initial phase where the model processes the entire user input prompt, in tokens/s), decode speed (subsequent phase where the model generates the output response one token at a time, in tokens/s), time to first token (TTFT) in milliseconds (ms), and memory usage (if accessible).

Table 3. Specifications of Test Devices

Device	Architecture	OS	RAM (GB)	Processor
Windows	x86_64	Windows 11	32	Intel Core i7, 14 cores
Android TV	armv7_32	Android	3	ARM Cortex-A55, 4 cores
Smartphone	armv8_64	Android	12	ARM Cortex-A55 + Cortex-A72 (4+4 cores)

Table 4. Compatibility of inference engines across Android architectures

Engine	armv7a (32-bit)	arm64-v8a (64-bit)
MLLM		X
MediaPipe		X
Llama.cpp	X	X
MNN	X	X
MLC-LLM		X

By implementing both deployment feasibility and software optimization perspectives, the proposed pipeline constitutes a reusable framework to guide the integration of generative language models in real-world embedded scenarios. Its modular structure enables straightforward adaptation to diverse hardware platforms and model configurations, positioning it as a facilitator for researchers and practitioners engaged in edge AI deployment.

3. Results and Discussion

The proposed evaluation is structured into two sequential experimental phases, aligned with the evaluation pipeline primarily introduced. The performance of each model was evaluated using both qualitative and quantitative metrics. For qualitative assessment, we examined response relevance, factuality, and the absence of hallucinations. Quantitative evaluation encompassed TTFT, generation speed, and total response time.

3.1. Feasibility Screening on Resource-Constrained Device (RQ1)

The initial phase of the study focused on identifying which SLM are capable of operating within the constraints of a highly resource-limited Android TV environment. A total of 21 SLMs were evaluated on the target device using 4-bit quantization. This phase yielded a critical insight: the majority of models failed to execute successfully, highlighting that deployment feasibility remains the primary barrier to on-device inference. Based on their observed runtime behavior, the models were classified into three distinct outcome categories, as described below:

- Category I - Functional Models: A total of nine models (M1–M9 in Table 1) were successfully loaded and executed full inference cycles on the target Android TV, enabling comprehensive quantitative evaluation. All selected models were quantized to 4 bits, as preliminary tests revealed that 8-bit alternatives offered no consistent performance gains while significantly increasing memory consumption, leading to higher failure risks.

- Category II - Models with Operational Failure: A second group of models (M10–M15 in Table 1) loaded successfully but failed during inference. Failure modes included repetitive generation loops, incoherent token sequences, and non-terminating generation, rendering them unsuitable for the target application.
- Category III - Models with Critical Loading Failures: The final category includes models that failed to initialize (M16–M21 in Table 1). These failures were predominantly caused by the application terminating upon load, which is consistent with the device exhausting its available RAM. This suggests a practical upper limit for model size on this class of hardware, with models approaching or exceeding 1 billion parameters presenting a high risk of failure.

3.2. Performance and Stability Trade-offs in Viable Models (RQ2)

For the nine models that were successfully executed (Category I), a detailed performance analysis was conducted using key metrics such as TTFT, generation speed (measured in tokens/s), and total response time. The results, presented in Table 5, reveal distinct performance profiles across models and expose critical trade-offs between inference speed, output quality, and response stability.

Table 5. Consolidated metrics for functional models. Response quality and consistency are measured qualitatively. TTFT and total generation time are measured quantitatively

Model	Response Quality	Tokens/s	Total Time (s)	TTFT (ms)	Consistency
InternVL3-1B	Excellent	1.15	129.6	51,483.0	High
Qwen2.5-0.5B	Excellent (Concise)	0.63	68.8	34,926.2	High
MiniCPM4-0.5B	Excellent (Concise)	0.55	122.0	54,327.6	Very High
HyperCLOVAX-0.5B	Good	0.57	141.6	65,235.0	Very High
SmolLM2-360M	Good (Hallucination)	0.76	181.0	41,603.4	High
Pythia410m-Instruct	Fair	0.67	130.4	45,798.8	Medium
OpenELM-450M-Instruct	Fair (Unstable)	0.65	160.0	68,136.5	Medium
SmolLM2-135M	Risk (Hallucination)	1.64	62.6	29,344.6	Low
Lite_Oute1-65M	Risk (Hallucination)	1.73	16.0	4,730.0	Low

The analysis identified recurring model archetypes that exhibit consistent patterns in performance, output quality, and runtime stability under constrained conditions. High-throughput models such as Lite_Oute1-65M and SmolLM2-135M achieved the highest generation speeds (1.73 and 1.64 tokens/s, respectively), but displayed higher variance and a greater tendency to generate non-factual content. In contrast, high-consistency models, including MiniCPM4-0.5B and HyperCLOVAX-0.5B, produced the lowest generation speeds yet demonstrated highly stable and predictable performance. Finally, balanced models such as InternVL3-1B and Qwen2.5-0.5B achieved a favorable trade-off between performance and response quality, with Qwen2.5-0.5B showing notable efficiency due to concise outputs and InternVL3-1B delivering high-quality responses with competitive generation speed for its size.

3.3. The Primacy of the Inference Engine for Acceleration (RQ3)

Having established the characterization baselines, the second phase of the study examined software-level strategies for accelerating inference across multiple hardware platforms. A representative model, SmolLM360-8bit, was used to ensure consistency throughout the comparative evaluation. The evaluation began with an analysis of the impact of OpenBLAS, a library specifically optimized for linear algebra operations on CPUs. As shown in Table 6, OpenBLAS provided a significant performance boost on the Windows x86

machine, doubling the prompt evaluation speed from 92.5 to 185 tokens/s. However, when deployed on the ARM-based Android devices (Smartphone and TV), it yielded a negligible improvement. This suggests that optimizations tailored for high-core-count x86 architectures do not readily translate to the low-power, few-core ARM processors common in edge devices.

Table 6. Evaluation results (in tokens/s) across different devices and BLAS configurations

Device	BLAS	Prefill Speed (tokens/s)	Decode Speed (tokens/s)	Total Time (s)
Windows	Disabled	92.52	14.56	18.14
Windows	Enabled	185.03	14.81	19.59
AndroidTV	Disabled	5.37	1.60	0.065
AndroidTV	Enabled	5.40	1.49	0.062
Smartphone	Disabled	11.31	7.96	21.62
Smartphone	Enabled	11.12	8.85	21.62

Table 7. Inference performance comparison between llama.cpp and MNN engines

Device	Engine	Prefill Speed (tokens/s)	Decode Speed (tokens/s)
Windows	llama.cpp	92.52	14.56
Smartphone	llama.cpp	11.31	7.96
AndroidTV	llama.cpp	5.37	1.60
Windows	MNN	156.61	96.54
Smartphone	MNN	80.64	22.54
AndroidTV	MNN	21.12	19.34

Subsequently, the performance of two distinct inference engines (llama.cpp and MNN) was compared across multiple hardware platforms. As presented in Table 7, the results demonstrate that the choice of inference engine constitutes a critical factor influencing on-device performance. On the Android TV, switching from llama.cpp to MNN increased prefill speed from 5.37 to 21.12 tokens/s (a **3.9x** improvement) and, most strikingly, boosted decode speed from 1.6 to 19.34 tokens/s (a **12x** improvement). Similar, gains were observed on the smartphone, improving prefill speed from 11.31 to 80.64 tokens/s. This finding highlights that for ARM-based edge devices, optimizing the software stack by selecting a highly efficient, architecture-aware inference engine can yield order-of-magnitude performance gains, far surpassing those from other attempted optimizations.

The superior performance observed with MNN on the 32-bit ARM-based TV platform can be explained by a combination of implementation and architecture level factors. MNN is designed as a lightweight mobile inference engine and provides optimized low-level kernels for ARM and x86 CPUs, including assembly-level optimizations intended to better exploit the characteristics of mobile processors [Jiang et al. 2020a, Wang et al. 2024]. In addition, its quantization pipeline reduces both computational cost and memory footprint by converting floating-point operations to lower-precision integer computations, such as INT8, which is particularly relevant for memory-constrained embedded devices. These aspects are important in 32-bit Android environments, where the

available virtual address space and memory bandwidth are more limited than in desktop or 64-bit mobile platforms. Therefore, the observed gains are likely related not only to raw arithmetic throughput, but also to reduced memory pressure, tensor reuse, and better cache behavior during inference.

In contrast, the limited impact of OpenBLAS on the ARM Cortex-A55 platform can be associated with the fact that BLAS acceleration mainly benefits large and regular matrix operations, while LLM inference involves a sequence of smaller, quantized, and memory-bound operations whose performance is strongly affected by cache locality, threading overhead, and runtime scheduling. This also explains why OpenBLAS produced more expressive gains on x86, where wider SIMD extensions and stronger multicore execution can be more effectively exploited. Thus, although these results should not be interpreted as a universal property of all 32-bit ARM devices, they provide evidence that MNN is better aligned with the constraints of the evaluated TV architecture.

4. Conclusion

Following the performance evaluation phase, a post-mortem analysis was conducted to investigate failure cases and better understand the trade-offs between inference speed and output quality in constrained edge environments. The proposed pipeline assessed the deployment of SLMs on edge devices using Android TV as a representative case of constrained consumer hardware, through a two-phase study evaluating both the feasibility of quantized models and the impact of software-level acceleration strategies on inference efficiency. Empirical results revealed that several models, including SmolLM2-135M and LiteOute-165M, achieved high inference speeds but frequently generated hallucinated or incoherent responses, demonstrating that low latency alone is insufficient for reliable deployment in user-facing applications. Conversely, larger models such as InternVL3-1B showed performance plateaus, where increased parameter counts did not lead to meaningful improvements in output quality or robustness, suggesting that model scaling is often constrained by memory, computational limitations, and optimization compatibility on edge hardware.

The experiments also showed that most models above 500 million parameters were unsuitable for the tested environment due to loading failures or unstable inference behavior, whereas a subset of 4-bit quantized SLMs, particularly MiniCPM4-0.5B and Qwen2.5-0.5B, demonstrated stable execution and acceptable response quality under strict resource constraints. Furthermore, strong sensitivity to software and hardware configurations was observed: while CPU-level libraries such as OpenBLAS provided minimal improvements on ARM architectures, replacing llama.cpp with the MNN inference engine resulted in up to $12\times$ faster token generation, with only two engines meeting the requirements of 32-bit devices. These findings indicate that performance in edge AI systems emerges from the interaction between model architecture, inference engine, and hardware platform, providing practical guidance for deploying generative AI in resource-constrained environments and motivating future work on runtime optimizations, adaptive multi-model loading strategies, conversational memory mechanisms, and multimodal or speech-based interfaces.

We acknowledge that exhaustive evaluations on additional devices are required for generalization purposes, especially in similar TV architectures. Nevertheless, as shown

in Tables 6 and 7, the MNN inference engine achieved lower latency compared with other architectures, such as Smartphone (ARMv8) and Windows (x86_64), which provides initial evidence that this behavior may remain consistent across similar platforms. As future work, we propose expanding the experiments to other similar TV devices, evaluating more recent SLMs, extending the pipeline to compact vision-language models (VLMs), exploring additional inference engines, such as OpenVINO, TensorRT, and ExecuTorch, and investigating personalized pruning and quantization methods.

References

Agrawal, R., Kumar, H., and Lnu, S. R. (2025). Efficient llms for edge devices: Pruning, quantization, and distillation techniques. In *2025 International Conference on Machine Learning and Autonomous Systems (ICMLAS)*, pages 1413–1418. IEEE.

Allal, L. B., Lozhkov, A., Bakouch, E., Blázquez, G. M., Penedo, G., Tunstall, L., Marafioti, A., Kydlíček, H., Lajarín, A. P., Srivastav, V., et al. (2025). Smollm2: When smol goes big–data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*.

Bhat, A., Mondal, A., and Tripathy, A. (2024). Llm agents for internet of things (iot) applications. *CS598 JY2—Topics in LLM Agents; University of Illinois: Urbana, IL, USA*.

Biderman, S., Schoelkopf, H., Anthony, Q. G., Bradley, H., O'Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, U. S., Raff, E., et al. (2023). Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Cheng, D., Gu, Y., Huang, S., Bi, J., Huang, M., and Wei, F. (2024). Instruction pre-training: Language models are supervised multitask learners. *arXiv preprint arXiv:2406.14491*.

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., et al. (2024). Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

Dakle, P. P., Rallabandi, S., and Raghavan, P. (2022). Understanding bloom: An empirical study on diverse nlp tasks. *arXiv preprint arXiv:2211.14865*.

de Souza, I. M., Bandeira, F. I., Xavier, D., and de Lima Filho, E. B. (2025). Low memory consumption architecture for tv platform software modules. In *2025 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–4.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 con-*

ference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers), pages 4171–4186.

Du, Y., Sun, Z., Wang, Z., Chua, H., Zhang, J., and Ong, Y.-S. (2025). Active large language model-based knowledge distillation for session-based recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 11607–11615.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. (2024). The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Elhanashi, A., Dini, P., Saponara, S., and Zheng, Q. (2024). Advancements in tinyml: Applications, limitations, and impact on iot devices. *Electronics*, 13(17):3562.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., et al. (2024). The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Hillier, D., Guertler, L., Tan, C., Agrawal, P., Ruirui, C., and Cheng, B. (2024). Super tiny language models. *arXiv preprint arXiv:2405.14159*.

Jiang, X., Wang, H., Chen, Y., Wu, Z., Wang, L., Zou, B., Yang, Y., Cui, Z., Cai, Y., Yu, T., et al. (2020a). Mnn: A universal and efficient inference engine. *Proceedings of Machine Learning and Systems*, 2:1–13.

Jiang, X., Wang, H., Chen, Y., Wu, Z., Wang, L., Zou, B., Yang, Y., Cui, Z., Cai, Y., Yu, T., Lv, C., and Wu, Z. (2020b). Mnn: A universal and efficient inference engine. In *MLSys*.

Li, X., Lu, Z., Cai, D., Ma, X., and Xu, M. (2024). Large language models on mobile devices: Measurements, analysis, and insights. In *Proceedings of the Workshop on Edge and Mobile Foundation Models*, pages 1–6.

Liu, Z., Zhao, C., Iandola, F., Lai, C., Tian, Y., Fedorov, I., Xiong, Y., Chang, E., Shi, Y., Krishnamoorthi, R., et al. (2024). Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. *arXiv preprint arXiv:2402.14905*.

Mehta, S., Sekhavat, M. H., Cao, Q., Horton, M., Jin, Y., Sun, C., Mirzadeh, I., Najibi, M., Belenko, D., Zatloukal, P., and Rastegari, M. (2024). OpenELM: An Efficient Language Model Family with Open Training and Inference Framework. *arXiv.org*.

Qin, R., Liu, D., Xu, C., Yan, Z., Tan, Z., Jia, Z., Nassereldine, A., Li, J., Jiang, M., Abbasi, A., et al. (2024). Empirical guidelines for deploying llms onto resource-constrained edge devices. *ACM Transactions on Design Automation of Electronic Systems*.

Sai, S., Prasad, M., Dashore, G., Chamola, V., and Sikdar, B. (2024). On-device generative ai: the need, architectures, and challenges. *IEEE Consumer Electronics Magazine*.

Sanh, V. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. In *Proceedings of Thirty-third Conference on Neural Information Processing Systems (NIPS2019)*.

Sarah, A., Nittur Sridhar, S., Szankin, M., and Sundaresan, S. (2025). Llama-nas: Efficient neural architecture search for large language models. In *European Conference on Computer Vision*, pages 67–74. Springer.

Team, G., Kamath, A., Ferret, J., Pathak, S., Vieillard, N., Merhej, R., Perrin, S., Matejovicova, T., Ramé, A., Rivière, M., et al. (2025). Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

Wang, Z., Yang, J., Qian, X., Xing, S., Jiang, X., Lv, C., and Zhang, S. (2024). Mnn-llm: A generic inference engine for fast large language model deployment on mobile devices. In *Proceedings of the 6th ACM International Conference on Multimedia in Asia Workshops*, pages 1–7.

Yoo, K. M., Han, J., In, S., Jeon, H., Jeong, J., Kang, J., Kim, H., Kim, K.-M., Kim, M., Kim, S., et al. (2024). Hyperclova x technical report. *arXiv preprint arXiv:2404.01954*.

Zafrir, O., Boudoukh, G., Izsak, P., and Wasserblat, M. (2019). Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pages 36–39. IEEE.

Zhu, J., Wang, W., Chen, Z., Liu, Z., Ye, S., Gu, L., Tian, H., Duan, Y., Su, W., Shao, J., et al. (2025). Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv preprint arXiv:2504.10479*.