Unlocking the Edge deployment and ondevice acceleration of multi-LoRA enabled one-for-all foundational LLM

Sravanth Kodavanti[†,*,✉], **Sowmya Vajrala**[†,*], **Srinivas Miriyala**[†,*]
Utsav Tiwari[†], **Uttam Kumar**[†], **Utkarsh Kumar Mahawar**[†], **Achal Pratap Singh**[†]
Arya D[†], **Narendra Mutyala**[†], **Vikram Nelvoy Rajendiran**[†], **Sharan Kumar Allur**[†]
Euntaik Lee[‡], **Dohyoung Kim**[‡], **HyeonSu Lee**[‡], **Gyusung Cho**[‡], **JungBae Kim**[‡]
[†] Samsung Research Institute Bangalore, India, [‡] Samsung Electronics, Suwon, South Korea
[*] Equal Contribution
[✉] Corresponding author: k.sravanth@samsung.com

Abstract

Deploying large language models (LLMs) on smartphones poses significant engineering challenges due to stringent constraints on memory, latency, and runtime flexibility. In this work, we present a hardware-aware framework for efficient on-device inference of a LLaMA-based multilingual foundation model supporting multiple use cases on Samsung Galaxy S24 and S25 devices with SM8650 and SM8750 Qualcomm chipsets respectively. Our approach integrates application-specific LoRAs as runtime inputs to a single frozen inference graph, enabling dynamic task switching without recompilation or memory overhead. We further introduce a multi-stream decoding mechanism that concurrently generates stylistic variations - such as formal, polite, or jovial responses - within a single forward pass, reducing latency by up to 6×. To accelerate token generation, we apply Dynamic Self-Speculative Decoding (DS2D), a tree-based strategy that predicts future tokens without requiring a draft model, yielding up to 2.3× speedup in decode time. Combined with quantization to INT4 and architecture-level optimizations, our system achieves 4-6× overall improvements in memory and latency while maintaining accuracy across 9 languages and 8 tasks. These results demonstrate practical feasibility of deploying multi-use-case LLMs on edge devices, advancing the commercial viability of Generative AI in mobile platforms.

1 Introduction

Deploying Large Language Models (LLMs) on mobile devices promises significant benefits in privacy, latency, and offline accessibility. However, bringing the flexibility of server-scale LLM adaptation-especially via Parameter-Efficient Fine-Tuning (PEFT) methods like LoRA-into the constrained environment of on-device inference presents several non-trivial challenges.

Unlike server-side systems, where models can be fine-tuned, recompiled, and dynamically loaded, on-device deployment must operate with frozen inference graphs, minimal memory, and strict runtime performance guarantees. Bridging the disconnect between flexible model development and the immutability of on-device inference demands a fundamental shift in how adaptable LLMs are engineered for real-world deployment.

In this work, we present a practical framework for real-time, multilingual, multi-use case LLM inference on Qualcomm's Neural Processing Unit (NPU) on SM8650 and SM8750 chipsets deployed on commercial smartphones, specifically the Samsung Galaxy S24 and S25. Our solution builds on a quantized LLaMA-based model integrated with eight application-specific LoRA modules across nine languages. Unlike conventional LoRA usage, where models are statically merged during training, we treat LoRAs as runtime inputs to the frozen inference graph, enabling dynamic task switching without recompilation or re-quantization. This design significantly reduces storage requirements and supports plug-and-play use cases on embedded device.

To meet hardware efficiency constraints, we perform targeted architectural transformations, including converting multi-head attention into parallel single-head paths, reparameterizing linear layers into convolutions, and quantizing both activations and weights to 4-bit precision (INT4) while utilizing mixed precision strategies to balance accuracy. These optimizations achieve substantial improvements in memory and latency while maintaining accuracy within acceptable limits.

Beyond architectural and quantization enhancements, we introduce a multi-stream token generation strategy that enables simultaneous inference across stylistic variants within a single decoding pass. For use cases such as tone transfer or stylistic rewriting (e.g., converting input into polite, formal,

Findings of the Association for Computational Linguistics: ACL 2026, pages 42443–42455

or jovial responses), traditional inference would require eight separate decoding iterations-one per style. By exploiting the fact that all use cases share the same frozen graph and memory layout, and that stylistic variants often are driven by the first token, we design a masked decoding scheme that modifies the first-token sampling and allows concurrent generation of eight distinct outputs over a shared KV-cache and tensor layout. This yields up to 6× latency and memory reduction for stylistic generation without any change to the model binary or graph.

Finally, to address decode-time latency bottlenecks, we propose a speculative sampling strategy based on tree-based branching with prefix tuning. Our method operates without needing a separate draft model or retraining, making it fully compatible with frozen, single-graph inference pipelines. This enables semi-autoregressive decoding with up to 2× improvement in tokens-per-second throughput, further pushing the boundary of what's achievable in mobile LLM deployment without reliance on cloud infrastructure.

2 Reported Works

The Transformer architecture has played a pivotal role in the advancement of generative AI, enabling large language models (LLMs) like LLAMA (Touvron et al., 2023), GPT (Achiam et al., 2023) to produce high-quality, human-like text and opening up new possibilities for applications such as language translation, text summarization, and content generation. However, the increasing size and complexity of these models have also raised concerns about their computational efficiency and environmental sustainability. To address these challenges, researchers have proposed various techniques to optimize LLMs for deployment on resource-constrained devices.

Recent works have focused on developing efficient LLMs that can be deployed on mobile and edge devices. MobiLlama (Thawakar et al., 2024) is one such model that has gained significant attention due to its compact size and competitive performance.

Quantization, on the other hand, is a widely used technique to reduce the precision of model weights and activations, thereby reducing memory footprint and computational requirements. QLoRA (Dettmers et al., 2023) and QaLoRA (Xu et al., 2023) quantization-based low-rank adaptation techniques that have shown promising results in reducing the precision of LLMs while maintaining their performance.

Low-Rank Adaptation (LoRA) (Hu et al., 2022) is a technique used to adapt large language models (LLMs) to specific tasks or domains by applying low-rank matrices to the model weights. Quantization, on the other hand, is a widely used technique to reduce the precision of model weights and activations, thereby reducing memory footprint and computational requirements. QLoRA (Dettmers et al., 2023) and QaLoRA (Xu et al., 2023) are two recent quantization-based low-rank adaptation techniques that have shown promising results in reducing the precision of LLMs while maintaining their performance.

These techniques have been applied to various NLP tasks, including language modeling and text classification.

In addition to quantization, model optimizations like Flash Attention (Dao et al., 2022) have been proposed to reduce the computational overhead of self-attention mechanisms in transformers. Flash Attention achieves this by using a novel attention mechanism that reduces the number of computations required.

Apart from efficient attention, Speculative decoding is another prominent technique used to accelerate the inference speed of LLMs by predicting the output tokens in parallel. Eagle (Li et al., 2024), Medusa (Cai et al., 2024), and BiTA (Lin et al., 2025) are recent works that have proposed novel speculative decoding methods to improve the inference speed of LLMs.

Despite the advancements in efficient large language models (LLMs), quantization techniques, and speculative decoding methods, deploying a single LLM for multiple tasks on resource-constrained devices remains a significant challenge. Even efficient models like MobiLlama are not sufficient to handle multiple tasks across different languages. Furthermore, optimizations like Flash Attention are not compatible with Neural Processing Units (NPUs), and draft-based speculative decoding methods require additional memory, which is a scarce resource on constrained devices. It is essential to develop a one-for-all foundation model along with effective optimization methods to facilitate its deployment on resource-constrained devices.

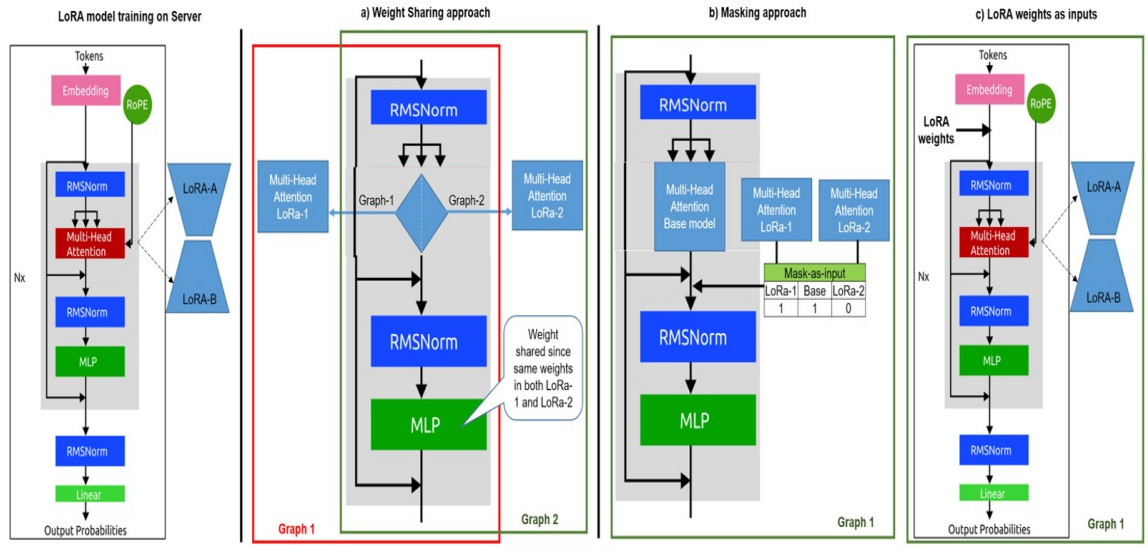

Figure 1: Proposed method for enabling multiple LoRA with a single LLM on embedded device.

3 Proposed Method

This work outlines the process starting from a trained foundation model with LoRAs, optimizing and deploying it on smartphones for no-cloud, fully on-device execution. It focuses on multi-LoRA enablement, inference optimizations, a novel lossless strategy for concurrent token generation, and speculative decoding for decode time acceleration. The authors refrain from describing the details of training the proprietary foundation model and LoRAs and solely focus on the innovations curated to ensure the on-device deployment. The readers are encouraged to refer to the seminal works on LLAMA (Touvron et al., 2023) and LoRA (Hu et al., 2022), which inspired the authors to build their foundation models.

3.1 LLM architecture and LoRA updates

The authors experiment with 1-billion and 3-billion parameter LLAMA-based LLM models, employing a decoder-only architecture with D sequential decoders. Tokenization generates embeddings of size E, which are transformed into Query (Q), Key (K), and Value (V) through linear layers, RMS normalization, and Rotary Position Embedding (RoPE) applied to Q and K. Each decoder block includes a multi-head attention (MHA) sub-block with H heads, followed by RMS normalization and an MLP block. The MHA output map (O) is processed by a learnable linear layer, producing Q, K, V, and O projection matrices with dimensions E x L and L x E, respectively, where L is the latent dimension.

In this work, LoRA weight updates are applied to these projection matrices (Eqs. 1–4), targeting the attention block. For each LoRA-supported layer, two parallel layers—LoRA-A and LoRA-B—are added. LoRA-A has dimensions E x d and d x L for Q, K, V, while LoRA-B has dimensions L x d and d x E for O projection, where d represents the LoRA dimension (rank). A scale factor s is introduced to control the impact of LoRA weights. In Eqs. 1 to 4, W denotes the projection weight matrix, and A and B represent the LoRA weights.

$$Y^Q = QW^Q + Q(sA^QB^Q) \quad (1)$$

$$Y^K = KW^K + K(sA^KB^K) \quad (2)$$

$$Y^V = VW^V + V(sA^VB^V) \quad (3)$$

$$Y^O = OW^O + O(sA^OB^O) \quad (4)$$

3.2 Multi-LoRA enablement on device

An overview of the proposed framework for LoRA enablement and runtime switching is presented in Figure 1. The framework has three approaches with various pros and cons.

In the first one (see Figure 1a), different graphs are prepared for different LoRA enabled tasks and the weights of the layers where LoRA is not applied, are shared between the graphs. During runtime, the corresponding graph is triggered. This ensures saving a significant amount of memory.

The second approach (see Figure 1b) maintains a single graph with multiple LoRAs and facilitates

dynamic switching between different tasks by applying one-hot encoded mask on the LoRAs. This approach provides inference benefit but increases the memory footprint.

The third approach (see Figure 1c) provides both latency and memory benefits by creating the foundation LLM graph with placeholders for LoRA weights and passing the LoRA weights as inputs along with tokens, assuming that LoRA weights for all use cases are of same dimensions. This simple yet elegant solution successfully enabled multiple LoRAs on device with low latency and memory.

3.3 Model Optimizations for On-Device LLM

Deploying Large Language Models (LLMs) on edge devices necessitates rigorous architectural and computational optimizations to meet the constraints of memory, latency, and power consumption while preserving functional accuracy. This section discusses several key model-level transformations and compilation-aware optimizations that enable efficient inference of LLMs on-device, tailored for modern NPUs and embedded GPUs.

Attention Parallelism. Multi-head attention (MHA), a core component of transformer-based LLMs, presents challenges in hardware execution due to its sequential dependencies and shared projection layers. To facilitate parallel processing on hardware accelerators, we decompose the MHA into multiple single-head attention (SHA) operations. Each SHA can be independently mapped to a parallel core, improving throughput.

This fine-grained decomposition also enables head-wise scheduling and execution on heterogeneous compute units, aligning with the architectural strengths of NPUs designed for independent low-dimensional matrix operations.

Linear-to-Convolutional Layer Transformation. Another major optimization involves reinterpreting linear (dense) layers as 1x1 convolutions. While mathematically equivalent in functionality, this transformation allows the model to exploit highly optimized convolution kernels present in embedded hardware backends. Modern mobile NPUs and GPUs, originally optimized for vision tasks, offer superior support for convolutional operations via Winograd algorithms, tiling strategies, and pipelined execution units. By casting dense layers in the form of point-wise convolutions, we enable efficient reuse of existing high-performance operators, reducing both latency and energy overheads.

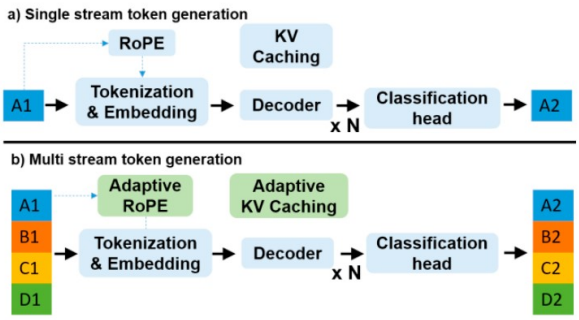

Figure 2: Schematic for Concurrent token generation

Constant Folding and Graph Simplification. During graph-level compilation, we apply constant folding to precompute operations involving static weights or configuration tensors. Examples include pre-multiplication of learned scale parameters in layer normalization or static reshaping and permutation of input dimensions. These algebraic simplifications reduce runtime computation and memory access, streamlining the execution graph. In combination with operator fusion—where adjacent linear and activation layers are merged into a single kernel—these techniques reduce kernel launch overhead and improve data locality in memory-constrained environments.

LLM Quantization Precision quantization plays a central role in enabling LLM inference on resource-constrained devices. In our framework, we employ Quantization-Aware Training (QAT) to simulate low-precision effects during training, allowing the model to learn robust representations under quantized conditions. We adopt a mixed-precision regime, where activations are quantized to per-tensor scaling for runtime simplicity to Int8 and weights per-channel to Int4, ensuring a trade-off between accuracy and compression.

3.4 Concurrent token generation (CTG)

To improve user experience, the LLM model generates multiple response options for personalization. For example, Smart-Reply provides 8 diverse prompts for users to choose from. However, this requires running the LoRA-enabled LLM sequentially 8 times, increasing latency by 8x. To address this, the proposed LoRA approaches enable handling multiple output streams simultaneously. The multi-stream token generation algorithm allows concurrent creation of n output streams, as shown in Figure 2. More elaborate details on the flowsheet of on-device execution and the mask used

for implementing the method are presented in the Appendix A.1.

3.5 Dynamic Self Speculative Decoding

Speculative decoding has emerged as a promising approach to overcome inference-time inefficiencies in LLMs, which typically suffer from sequential, memory-bound decoding constraints. Some techniques use draft models and additional heads but they incur additional memory, while some early exit based methods require retraining the whole model to make the intermediate layers predict proper tokens. To avoid these issues, we utilized Prefix tuning based parameter efficient mechanism to fine-tune the foundational LLM, such that it generates multiple tokens in addition to the auto regressive token in one step.

In this method inspired from BiTA (Lin et al., 2025) task-specific m forecast embeddings are appended to the prompt embedding. These additional m forecast embedding ensure that the model predicts 1+m tokens at a time, thus converting its nature from auto-regressive to semi-auto-regressive (SAR). While the first token is sampled from the frozen LLM's distribution, ensuring it to be same as the case without the forecast embedding, the remaining m tokens are generated with reduced fidelity, thus requiring verification during the inference. The fine-tuning procedure to generate SAR tokens is facilitated through the prefix tuning method (see Appendix A.2 for further details).

As illustrated in Figure 3, during the inference step 1, the forecast prefix (shape: p×d), prompt embeddings (shape: q×d), and forecast embeddings (shape: m × d) are passed as inputs to the Gauss L model. In the output logits, the token marked in orange is generated from the prompt and is referred to as the "last verified token,". For the logits derived from the forecast embeddings, a softmax operation is performed, followed by sampling to generate multiple draft tokens. In the example provided in Figure 3, with m = 2, i.e., 2 forecast embeddings, (3, 2) branch configuration is implemented. In inference step 1, the top 3 and top 2 tokens are sampled from the logits of the first and second forecast embedding, respectively. These draft tokens are used to build a tree-based structure, as depicted in Figure 3. In inference step 2, the generated draft tokens undergo verification along with new draft token generation. Verification is done by appending the 9 draft tokens to the prompt token, and new draft generation is done by appending a set of m forecast embeddings for each target and draft token. In this case, a total of 10 tokens and 20 forecast embeddings are passed as input. Based on the acceptance of the current draft tokens, the next draft tokens will be generated. In Figure 3, since tokens 1-3-5 are accepted, drafts are generated from the output logits of forecast embeddings corresponding to token 5. The on-device deployment of this dynamic selection is facilitated through a mask described in the Appendix A.2.

Optimal Branch Configuration. As shown in Figure 3, the input shape (mainly rows) during inference (step 2) for a branch configuration of 3,2 corresponds to p+q+30 tokens. Since KV Cache mechanism will be implemented the p forecast prefix and all the prompt tokens except the previous token will be cached. Thus, the number of rows in input is 30. Since these inputs will be processed in parallel by the Transformer architecture, they are analogous to the batch size used during training or inference. Owing to the optimized architecture design of NPU (or GPUs), we consider the input size to be 32 (powers of 2) to ensure maximum benefit. Given the input size as 32, we can try different branch configurations that can result in a total of 31 tokens in the input + 1 previous token. Among all those configurations, the optimal configuration for use-case is determined as the one which results in maximum acceptance rate. Since, the branch configuration can be determined optimally for

4 Experiments & Methods

4.1 Experiments on GS24 with 1B LLM

Table 1: On-Device Performance of LoRA weight sharing

	Base		**LoRA (2-tasks)**	
	1st Token	**Gen. model**	**1st Token**	**Gen. model**
Num. graphs	2		4	
Size	718 MB		1.1 GB	
Latency	45ms	22ms	45ms	22ms

Table 2: On-Device Performance of LoRA weight masking and LoRA as input approaches measured on 2 tasks

Approach	**Metric**	**Prefill Mode**		**Generative Mode**	
		Base	**LoRA**	**Base**	**LoRA**
Masking	Size	718 MB	894 MB	718 MB	894 MB
	Latency	45ms	75ms	22ms	30ms
LoRA as Input	Size	718 MB	686 MB	718 MB	686 MB
	Latency	45ms	52ms	22ms	25ms

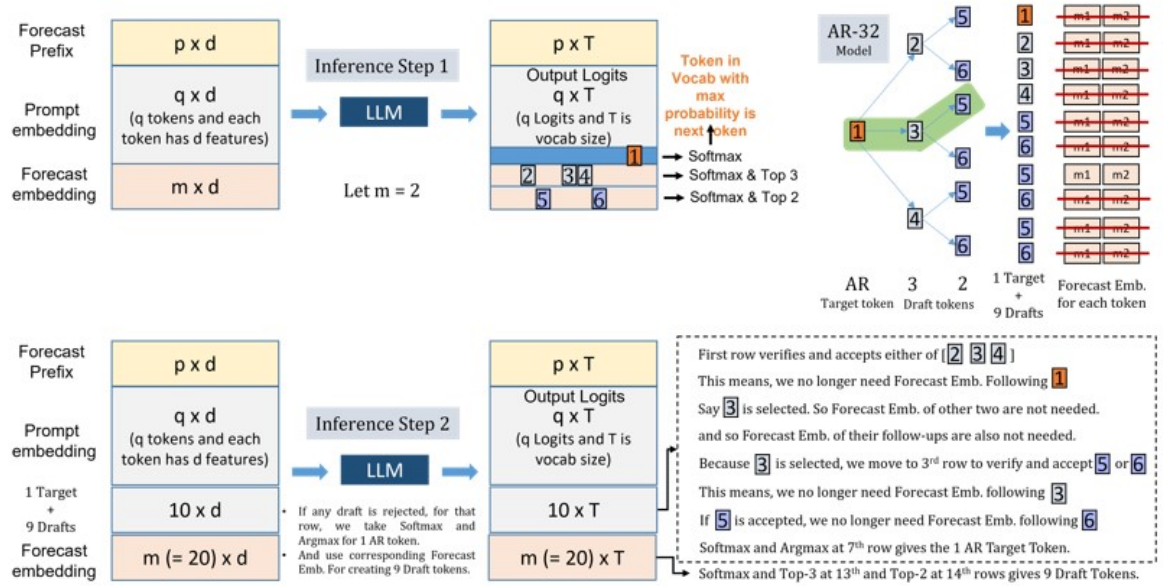

Figure 3: Approach for self-speculative decoding and the procedure for fine-tuning the model with Prefix tuning.

Table 3: Inference time analysis with CTG

Streams	Prefill Latency (ms)	AR Latency (ms)	Total Time (ms)	Formula
1	40	23	174	$(23 \times 8) + 40$
8	40	23	63	$23 + 40$

After quantizing the model (see Appendix A.3.1 for detailed results) and performing model optimization for its on-device deployment, the on-device performance was profiled on the GS24, with the proposed multi-LoRA method tested across the three approaches for enabling dynamic runtime switching. Results are presented in Tables 1, 2, showing that while all three approaches performed similarly for fewer use-cases (e.g., Table 2), the weight-as-input approach proved scalable for multiple use-cases.

Table 4: 1B LLM On-Device Accuracy Results (%) on GS24 Ultra.

Task	Korean	English	German	Spanish	French	Italian
Correction	96.83	97.60	90.20	95.90	95.37	94.58
Style	97.41	97.72	96.71	98.62	96.48	98.58
Smart Reply	103.40	111.13	97.36	96.02	97.37	100.79

The accuracy of various tasks and languages measured on-device using the quantized LLM and LoRAs with proposed post-training quantization (PTQ) approaches is detailed in Table 4.

We also evaluated the on-device accuracy of various tasks, such as correction, style, and Smart Reply, using the quantized LLM and LoRA with our proposed Post-Training Quantization (PTQ) approach. The results are presented in Table 4, which shows that our approach maintains accuracy while reducing latency. The one-for-all foundation model's overall memory and generation performance on-device is presented in Table 5. Exhaustive results and demonstrations are provided in Appendix A.3.1.

Table 5: On-device Performance of One-for-all LLM model

Memory and Initialization	
Peak Memory (MB)	967
Load Time (ms)	624.1
First Token Latency (ms)	155.9
Generation Performance	
Per-Token Latency (ms)	24.19
Token Generation Rate (tokens/sec)	41.32
Total End-to-End Time (sec)	2.85

4.2 Scaling to a 3B LLM on GS25 with DS2D

With the deployment of a 3B LLM on GS25, we further accelerated the model using our proposed Dynamic Self-Speculative Decoding (DS2D) approach. The implementation of DS2D, which chooses the optimal branch for each use-case, resulted in a 2-fold improvement in tokens per second compared to non-speculative decoding, as shown in Table 6. Table 7 presents the acceleration obtained with DS2D measured in average tokens per inference & tokens per second on GS25 Ultra, with optimal branch configurations enabling a single graph

Table 6: 3B LLM Performance for different use cases w/ & w/o DS2D on GS25 Ultra

Use Case	Prefill (ms)	Decode time (ms)		Tokens/sec		Peak Mem (MB)	Init (ms)
		w/o DS2D	w/ DS2D	w/o DS2D	w/ DS2D		
Correction	208.72	50.17	**22.3**	19.93	**44.84**	2463.44	1507.96
Composer	209.57	53.23	**28.57**	18.79	**35.00**	2468.51	1499
Style	209.34	50.42	**25.21**	19.83	**39.67**	2465.42	1534.27
Health	210.7	51.92	**28.77**	19.26	**34.76**	2468.52	1551.98
Summarization	797.22	52.91	**24.12**	18.90	**41.46**	2486.98	1576.78
Natural Language	209.8	51.87	**23.64**	19.28	**42.30**	2480.09	1547.54
ST Energy	210.77	52.76	**33.48**	18.95	**29.87**	2486.3	1487.4
AI Brief	404.88	52.85	**37.75**	18.92	**26.49**	2484.68	1509.68

for all use-cases. On-device accuracy comparisons between quantized and full-precision models across 9 languages are presented in Table 8. This improvement led to the real-world deployment of our approach in the Galaxy S25 family. Additional results included in appendix A.3.2.

Appendix Due to space constraints, the following details, extended experiments, and on-device demonstrations are provided in the appendix:

- **Concurrent token generation (CTG):** Pipeline and mask for Concurrent token generation.
- **Dynamic Self Speculative Decoding:** Details of finetuning LLM for SAR generation and mask used during inference.
- **Deploying a 1B LLM:** Comparison between full precision and quantized models, latency improvements with proposed graph optimizations and on device demonstrations of smart reply and style suggestion usecases.
- **Scaling to a 3B LLM:** Demonstrations on device for writing assist, summarization and health energy score use cases.

5 Conclusion

This work presents an effort to optimize the inference pipeline for deploying a single pre-trained foundation model to serve multiple Generative AI use-cases in the Galaxy AI feature of Samsung flagships. The proposed approach implements methods and optimizations for enabling multiple quantized low rank adapters capable of switching on demand at runtime. The empirical analysis demonstrates 8 different uses-cases with a single large language foundation model that results in 3 and 2 fold improvement in on-device memory and token rate, respectively.

The framework proposed in this research is first-of-its-kind which led to the successful on-device deployment of Generative AI-based language understanding use-cases. This framework will serve as a guide for best practices in optimizing the memory-intensive foundation models for efficient on-device deployment. The concepts such as the quantization, beyond the foundation models for Generative AI and apply to all kinds of AI models, resulting in a much larger scope, and efficient and sustainable on-device AI.

This work establishes a scalable and efficient approach for deploying LLMs fully on-device. Through model re-engineering, multi-LoRA handling and the development of novel Self-Speculative Decoding, we achieve real-time inference across multiple tasks and languages on mobile NPUs. Our framework successfully commercialized advanced LLM capabilities without cloud reliance, setting a new benchmark for edge AI.

6 Limitations

While our system achieves significant improvements in memory efficiency, latency, and multi-use-case deployment of LLMs on-device, it is subject to limitations which will be addressed in future works:

- LoRA Dimensions: Our LoRA-as-input design assumes that all LoRAs have identical dimensionality. This is necessary to maintain compatibility with the frozen inference graph and its fixed placeholder structure. Supporting heterogeneous LoRA sizes would require dynamic graph construction or runtime shape management, which is infeasible under current deployment constraints.
- Forecast Embeddings Are Static in DS2D: In our Dynamic Self-Speculative Decoding (DS2D) approach, the forecast embeddings used to precompute semi-autoregressive

Table 7: Acceleration obtained in 8 different use-cases under different branch configurations on GS25 with SM8750 Qualcomm chipset

Branch Config	Correction		Composer		Style		Health		Summarization		Gallery		ST Energy		AI Brief	
	tokens/inf	tokens/s	tokens/inf	tokens/s	tokens/inf	tokens/s	tokens/inf	tokens/s	tokens/inf	tokens/s	tokens/inf	tokens/s	tokens/inf	tokens/s	tokens/inf	tokens/s
15	1.7	39.26	**1.72**	**40.35**	1.7	39.27	**1.54**	**36.05**	1.76	40.82	**1.59**	**36.84**	**1.45**	**33.88**	**1.57**	**36.46**
1,8	2.05	45.88	1.51	35.47	1.81	41.38	1.31	30.69	1.7	39.07	1.49	34.45	1.08	25.49	1.32	30.91
2,3	2.14	47.69	1.56	36.57	1.94	43.85	1.37	31.98	1.74	39.8	1.63	37.32	1.21	28.51	1.45	33.44
3,2	2.13	47.94	1.6	37.61	**1.96**	**44.47**	1.41	33.09	**1.8**	**41.34**	1.58	36.38	1.25	29.35	1.5	34.74
4,1	2.02	45.31	1.62	37.83	1.85	42.05	1.48	34.5	1.77	40.37	1.52	35.15	1.3	30.31	1.48	34.09
1,1,5	2.25	49.44	1.39	32.32	1.84	41.51	1.24	28.99	1.58	35.98	1.35	31.27	1.07	25.13	1.27	29.33
1,2,2	**2.27**	**49.6**	1.43	33.24	1.92	43.13	1.25	29.1	1.67	37.98	1.43	32.86	1.07	25.02	1.31	30.25
2,1,1	2	44.55	1.49	34.36	1.8	40.58	1.32	30.48	1.64	37.43	1.46	33.39	1.21	27.92	1.43	32.92
1,1,1,2	2.14	46.95	1.37	31.57	1.81	40.47	1.24	28.69	1.53	33.82	1.35	29.56	1.07	23.87	1.26	28.39

Table 8: On-Device Accuracy measured for different use-cases in 9 languages on GS25. G-Eval measured with ChatGPT 4.5 as Evaluator Model and Relative Accuracy (RA) = $100 \times \frac{\text{OnDevice QAT Accuracy}}{\text{FP32 Accuracy}}$

Precision	Task/Language	Korean		English		German	
		G-Eval	RA	G-Eval	RA	G-Eval	RA
Fp32	Correction	2.53	NA	2.664	NA	2.35	NA
QAT - On Device		2.52	99.6047	2.7424	102.943	2.3892	101.668
Fp32	Style	2.94	NA	2.935	NA	2.806	NA
QAT - On Device		2.96	100.51	2.928	99.7445	2.871	102.309
Fp32	Smart Reply	2.7	NA	2.834	NA	2.621	NA
QAT - On Device		2.67	99.0724	2.81	99.1531	2.56	97.6726
Fp32	Summarization	2.71	NA	2.65	NA	2.65	NA
QAT - On Device		2.693	99.3542	2.688	101.446	2.641	99.66031
		Spanish		**French**		**Italian**	
		G-Eval	**RA**	**G-Eval**	**RA**	**G-Eval**	**RA**
Fp32	Correction	2.48	NA	2.64	NA	2.715	NA
QAT - On Device		2.535	102.22	2.608	98.769	2.635	97.053
Fp32	Style	2.958	NA	2.838	NA	2.743	NA
QAT - On Device		2.891	97.755	2.869	101.103	2.807	102.34
Fp32	Smart Reply	2.675	NA	2.595	NA	2.588	NA
QAT - On Device		2.65	99.065	2.68	103.276	2.63	101.62
Fp32	Summarization	2.647	NA	2.678	NA	2.502	NA
QAT - On Device		2.653	100.227	2.731	101.96	2.421	96.7619
		Portuguese		**Chinese**		**Japanese**	
		G-Eval	**RA**	**G-Eval**	**RA**	**G-Eval**	**RA**
Fp32	Correction	2.418	NA	2.24	NA	2.385	NA
QAT - On Device		2.638	109.078	2.174	97.031	2.305	96.6457
Fp32	Style	2.783	NA	2.929	NA	2.928	NA
QAT - On Device		2.836	101.919	2.878	98.228	2.91	99.392
Fp32	Smart Reply	2.534	NA	2.735	NA	2.534	NA
QAT - On Device		2.67	105.367	2.76	100.91	2.25	88.7924
Fp32	Summarization	2.647	NA	2.64	NA	2.646	NA
QAT - On Device		2.624	99.13109	2.189	82.9167	2.535	95.8042

(SAR) tokens remain static across inference steps. Since they are not conditioned on the evolving input sequence, this can lower the token acceptance rate, particularly for inputs with high semantic variance or long-term dependencies.

- Predefined Style Variants: The multi-stream decoding strategy relies on a fixed set of stylistic variants defined at compile time. While this enables concurrent generation, it does not support dynamically defined or user-specified styles without recompilation or memory duplication.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*.

T Dao, DY Fu, S Ermon, A Rudra, and C Flashattention Ré. 2022. Fast and memory-efficient exact attention with io-awareness. *URL https://arxiv.org/abs/2205.14135*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*.

Feng Lin, Hanling Yi, Yifan Yang, Hongbin Li, Xiaotian Yu, Guangming Lu, and Rong Xiao. 2025. Bita: Bi-directional tuning for lossless acceleration in large language models. *Expert Systems with Applications*, 279:127305.

Omkar Thawakar, Ashmal Vayani, Salman Khan, Hisham Cholakal, Rao M Anwer, Michael Felsberg, Tim Baldwin, Eric P Xing, and Fahad Shahbaz Khan. 2024. Mobillama: Towards accurate and lightweight fully transparent gpt. *arXiv preprint arXiv:2402.16840*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, Xiaopeng Zhang, and Qi Tian. 2023. Qa-lora: Quantization-aware low-rank adaptation of large language models. *arXiv preprint arXiv:2309.14717*.

A Appendix

A.1 Concurrent token generation (CTG)

Figure 4 illustrates the pipeline for concurrent token generation. It starts with a single Prefill model execution, followed by a sampler generating 8 distinct output tokens. These tokens trigger the generative model flow for 8 sentences. An 8-fold generative model processes 8 inputs simultaneously, yielding 8 outputs. The KV-cache is divided into 8 isolated segments, with masks ensuring each token stream interacts only with its Prefill context and corresponding segment (see Figure 5).

A.2 Dynamic Self Speculative Decoding

To enable the LLM to generate tokens semi-auto-regressively, Prefix Tuning is employed. The dataset can be generated from the frozen pre-trained LLM in auto-regressive manner and can be rearranged to facilitate the finetuning as shown in Figure 6. The forecast prefix induces the capacity of semi-auto regression into already trained and frozen LLM. Further the causal mask is modified to avoid the interaction of the forecast prefix with prompt token as shown in Figure 7. Since the m new tokens are of low-fidelity, these tokens need to be validated during the next inference step, while generating new draft tokens simultaneously.

Optimal Branch Configuration The configuration (3, 2) which results in 9 draft tokens, is dynamic and use-case specific, allowing the determination of optimal branching for each use-case. The dynamic selection is learnt during the fine-tuning step and facilitated during the inference step through the mask as shown in Figure 7. Note that, since the first inference step does not have to validate any draft tokens, the mask used in inference step 1 will be much simpler than the one described in 7.

A.3 Experiments

A.3.1 Deploying a 1B LLM on GS24

The first set of experiments focuses on deploying a 1B-parameter LLM on the GS24, utilizing the sm8650 chipset's Neural Processing Unit (NPU). The study spans 36 use-cases across 4 tasks—text style/tone transfer, spelling/grammar correction, smart reply, and health summarization—in 9 languages: Korean, English, German, Spanish, Italian, French, Dutch, Chinese, and Japanese. Four distinct LoRAs were developed for these use-cases, and the results were evaluated after quantization, optimization, and deployment on-device.

Memory comparisons between full-precision models and their quantized variants (with weights and activations in INT-4/INT-16 and LoRA weights

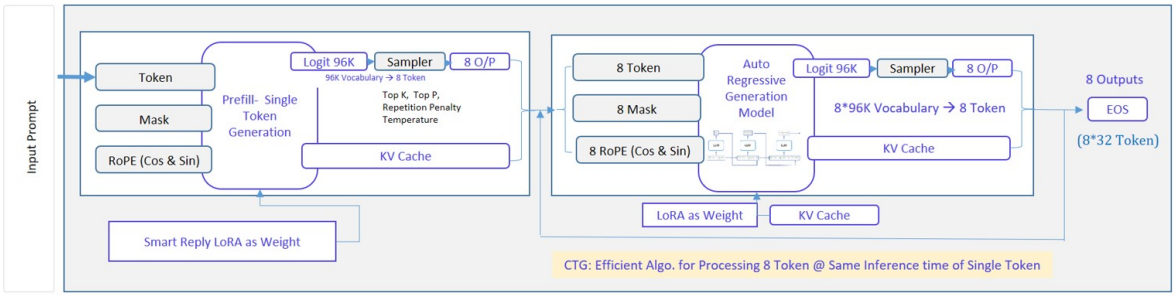

Figure 4: Flowsheet for the execution of Concurrent token generation (CTG) on device

Prefill	Mode	KV	Cache	S1			S2			S3			S4		T1	T2	T3	T4
Mask Representation for CTG																		
1	1	1	1	1											1			
1	1	1	1				1									1		
1	1	1	1							1							1	
1	1	1	1										1					1

Figure 5: Mask used during CTG. KV Cache is divided into 5 parts: one for the common prefill cache and four for individual sentences (S1 - S4). The tokens (T1 - T4) attend to the prefill and their respective sentence tokens.All empty values are 0's

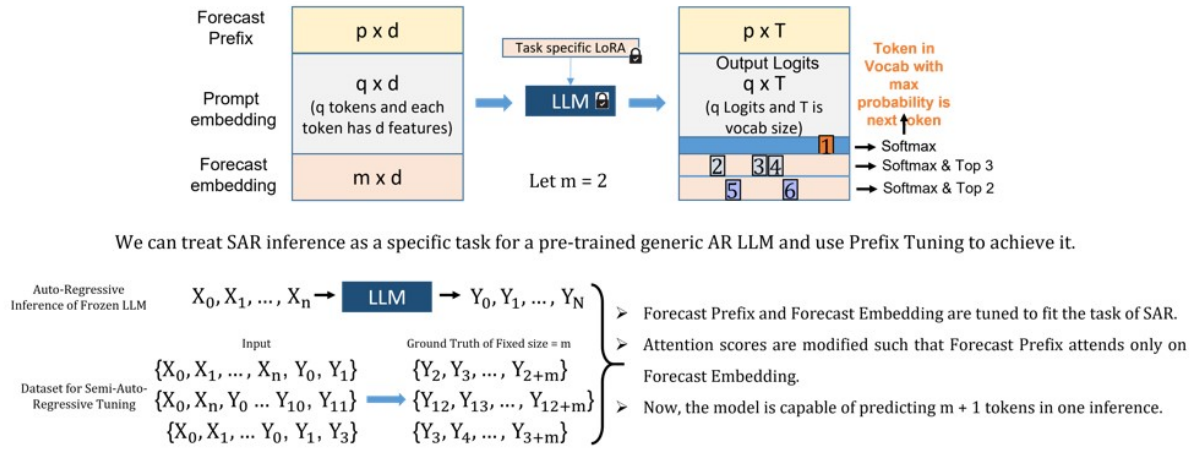

Figure 6: Approach for self-speculative decoding and the procedure for fine-tuning the model with Prefix tuning.

Table 9: Memory comparison between full precision and quantized models with LoRA for 4 tasks

Model	ROM (MB)
FP16 LLM + LoRA	1800+120
INT4 LLM + LoRA	600+120

Table 10: Performance improvement obtained with proposed Graph optimizations

Model-Type	Latency (ms)	
	First token	Generative model
Scalar-folding	45.8	19.385
Without Scalar-folding	47.3	20.51
LoRA-B splitting + Scalar folding	45.9	19.45
LoRA-B composite + Scalar folding	45.8	19.385
K-untransposed	52	23
K-transposed	45.8	19.385

in INT-16) are shown in Table 9 highlighting a 3x compression benefit from quantization. Graph optimizations, including scalar folding and LoRA multi-head splitting, are detailed in Table 10, with the latter improving quantization accuracy without affecting latency. The one-for-all foundation model's on-device performance is presented in Table 5.

The demonstration of User's inputs, corresponding prompts to the LLM, and the outputs generated by the LLM model with deployment of corresponding LoRAs is illustrated in Figure 8. The on-device

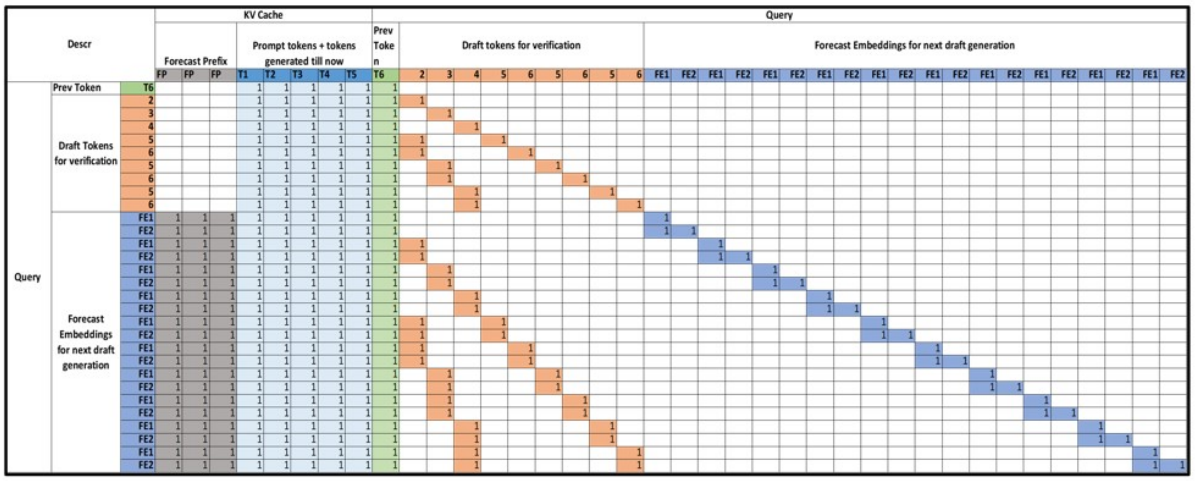

Figure 7: The mask used during inference step 2 in Figure 6. All the empty values indicate 0s.

	User's Input	Prompt / Input to LLM	Output to the User
Demonstration of Tone / Grammar Correction	Install all depenencies using your package manager.	"[{<(Task)>}]\nRemove all grammatical errors from this text\n\n[{<(Input)>}]\nInstall all depenencies using your package manager.\n\n[{<(Correction)>}]\n"	Install all dependencies using your package manager.
Demonstration of Style Change	(CNN)With the 42-kilometer-long Purple Line section of the Bengaluru Metro operational from Monday morning, delighted commuters took to X (formerly Twitter) to express their satisfaction with the relief from traffic congestion in the city and the availability of affordable travel. This newly operational line, which connects the East-West corridor in the city.	"[{<(Task)>}]\nChange a given user's input sentence or text to the 'Witty' style\n\n[{<(Input)>}]\n(CNN)With the 42-kilometer-long Purple Line section of the Bengaluru Metro operational from Monday morning, delighted commuters took to X (formerly Twitter) to express their satisfaction with the relief from traffic congestion in the city and the availability of affordable travel. This newly operational line, which connects the East-West corridor in the city.\n\n[{<(StyleTransfer)>}]\n"	The Bengaluru Metro's 42-kilometer-long Purple Line section is now operational, and commuters are thrilled. They've expressed their joy on Twitter, sharing how much they appreciate the relief from traffic congestion and the affordable travel options. This newly launched line connects the East-West corridor in the city
Demonstration of Smart Reply	Others: Oh, sorry. I completely forgot to text you back. Others: I attended a meditation retreat last weekend. User: A whole weekend of meditation? How was it? Others: It was trasnformative for sure, but also challenging.	[{<(Task)>}]\nGenerate an appropriate response of the User in the following conversation with the User and Others.\n\n[{<(Dialogue)>}]\nUser: Yo, where are you hiding?\\nUser: I haven't heard from you in over a week.\\nOthers: Oh, sorry. I completely forgot to text you back.\\nOthers: I attended a meditation retreat last weekend.\\nUser: A whole weekend of meditation? How was it?\\nOthers: It was trasnformative for sure, but also challenging.\n\n[{<(Reply)>}]\n	User: Sounds intense, can't wait to hear! User: Sounds intriguing, can't wait! User: Wow, sounds like a blast! User: Sounds intriguing, can't wait! User: Sounds intense, I missed it! User: Sounds intense! I'd love to try it. User: Sounds intense, I'll give it a shot. User: Sounds like a challenging one!

Figure 8: Demonstration of User's inputs, corresponding prompts to the LLM, and the outputs generated by the LLM model with deployment of corresponding LoRAs.

user inference for different tasks and languages was demonstrated on the GS24 and extended to the Samsung Galaxy Smartwatch via smart connectivity (Figures 9 & 10), showcasing the method's applicability across devices.

A.3.2 Scaling to a 3B LLM with DS2D

Building on the success of the 1B LLM deployment, the second set of experiments scales up to a 3B-parameter model deployed on the GS25 Ultra, leveraging DS2D for further acceleration. The proposed method was applied to all 8 use-cases of language understanding commercialized in GS25 (Qualcomm) and planned for Galaxy Flip/Fold 7 (LSI-Exynos).

Post-deployment demonstrations of LLM applications for writing assist, summarization and health energy score, and are shown in Figures 11 12 & 13 highlighting the method's real-world applicability.

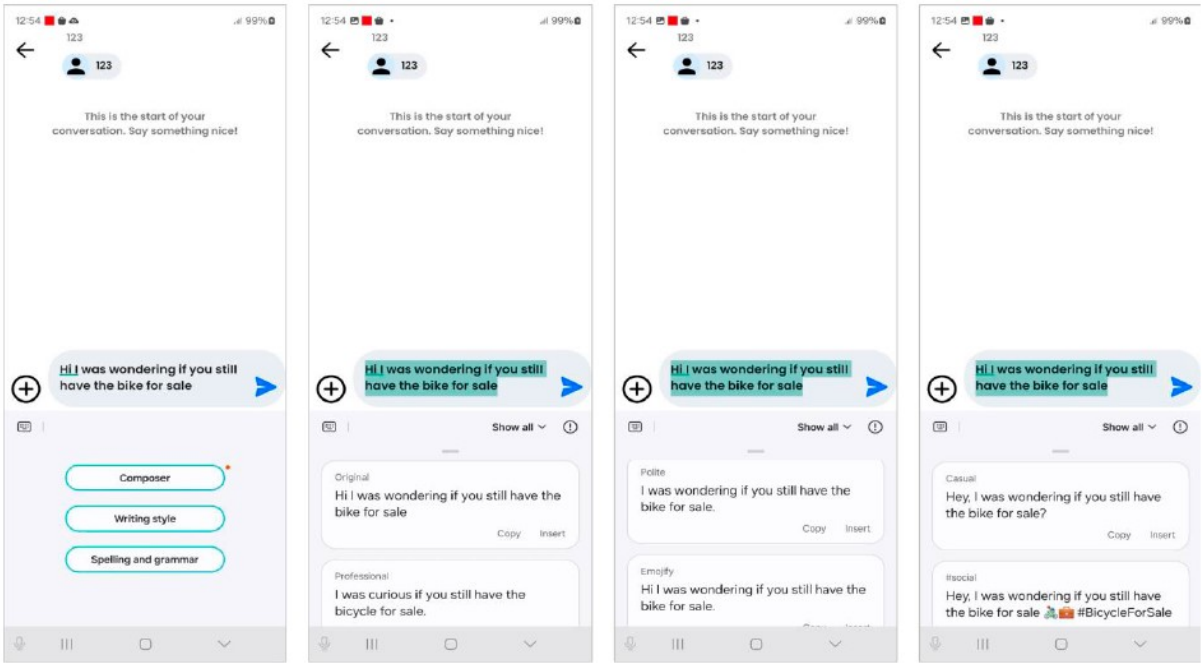

Figure 9: Demonstration of Style suggestion (Professional, Polite, Emojify, Casual & Social) on GS24 Ultra.

Figure 10: Demonstration of Smart Reply in Samsung Galaxy Smart Watch

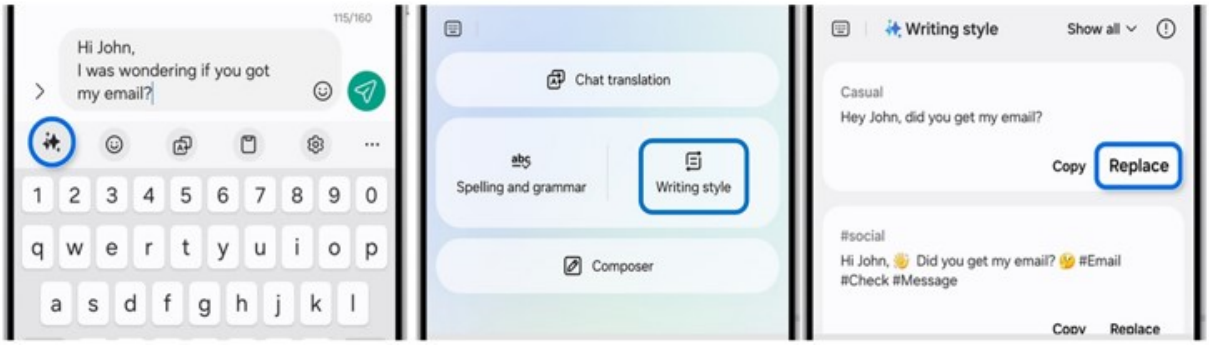

Figure 11: Demonstration of writing assist usecase in chat on Samsung GS25 Ultra Smart Phone

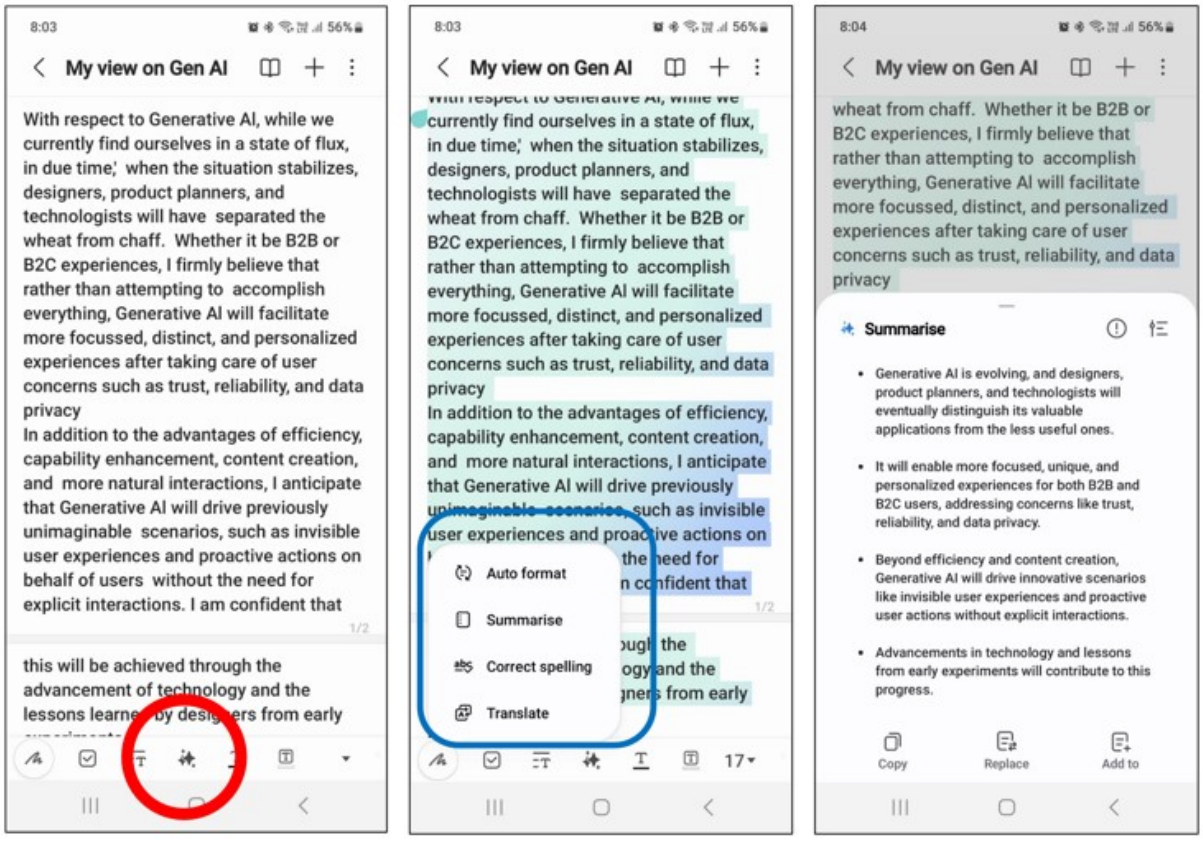

Figure 12: Demonstration of summarization usecase on Samsung GS25 Ultra Smart Phone

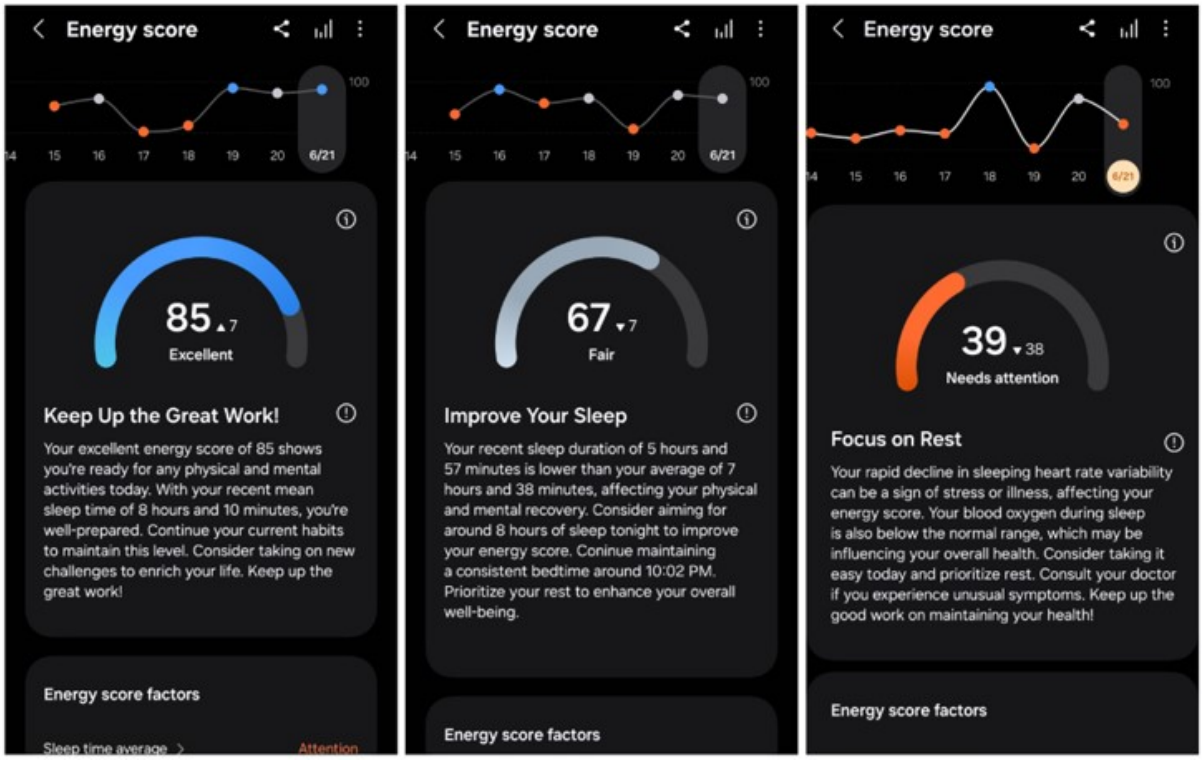

Figure 13: Demonstration of Energy score usecase in health application on Samsung GS25 Ultra Smart Phone