Memory-Efficient Edge Chatbot on Raspberry Pi Using Optimized KV-Cache Strategies

Sudman Sakib, Ian Penrod, Suhong Gu, Sai Kiran
Department of Electrical and Computer Engineering
The University of Texas at San Antonio

***Abstract*—This project implements memory-efficient inference of a Transformer-based chatbot (DistilGPT-2) deployed on a Raspberry Pi. We implement and evaluate three KV-cache optimization strategies—sliding-window, paged-turn, and quantized KV-cache—and measure their effects on latency, throughput, and memory consumption. Results demonstrate the trade-offs among cache strategies and highlight the practical constraints of deploying large language models on edge hardware.**

I. Introduction

Recent advances in Transformer-based language models have enabled interactive chatbots with strong contextual reasoning. Deploying these models on embedded devices, however, is challenging due to strict memory and compute limitations. The KV-cache mechanism plays a central role in enabling efficient autoregressive inference, but its memory footprint grows linearly with sequence length. This project investigates how cache-aware strategies can reduce memory usage and improve performance on constrained hardware.

II. System Overview

A. Hardware Specifications

A Raspberry Pi (model and version) was used for deployment. System characteristics include:

- CPU: Quad-core ARM Cortex-A72 (or model)
- RAM: 4GB (or version)
- Storage: 256GB SSD (USB attached)
- OS: Raspberry Pi OS (64-bit)

III. Background

A. KV-Cache in Transformer Inference

Transformers generate tokens autoregressively. For each new token, self-attention requires computing key/value (KV) representations over all previous tokens. The KV-cache stores previously computed keys and values to avoid recomputation, reducing complexity from quadratic to linear.

However, the KV-cache grows linearly with context length, creating a memory bottleneck on embedded devices. Cache-aware strategies are therefore essential to bound memory usage.

IV. Cache Strategies and Quantization Methods

A. Sliding-Window Cache

Maintains the most recent N tokens. Reduces prompt length during generation when multi-turn history grows. Lightweight and requires minimal bookkeeping.

B. Paged Cache

Stores conversation history in fixed-size “pages” (turns), discarding older turns once capacity is reached. Useful for structured dialogues where retention of several recent turns is necessary.

C. Quantized KV-Cache (INT8)

Compresses key/value tensors from floating-point to 8-bit integer form. Reduces KV-cache size but introduces additional quantization/dequantization operations each step. Most beneficial when KV-cache persists across turns.

D. Backend Modifications

A custom token-by-token generation loop was implemented to:

- expose KV tensors each decoding step,
- allow cache injection via `store()` and `get()`,
- support Pi-friendly CPU-only inference.

E. Baseline Model and Setup

A pretrained DistilGPT2 model was used for Phase 1. Inference was verified with no cache enabled, and baseline metrics (latency, tokens/sec, memory usage) were recorded.

F. Cache Strategies Implemented

1) **No Cache (Baseline)**: Full context, no trimming.
2) **Sliding-Window Cache**: Retains last N tokens, discards older ones.
3) **Paged Cache**: Maintains last K conversational turns.
4) **Quantized KV-cache (Optional)**: KV tensors compressed to INT8.

Each strategy is integrated into the inference loop as part of the CLI chatbot.

G. Benchmark Prompts

All cache variants were evaluated using identical prompts:

- “Explain Transformer architecture in neural network.”
- “What is a Raspberry Pi?”
- “Describe sliding-window vs paged cache vs quantized Cache.”

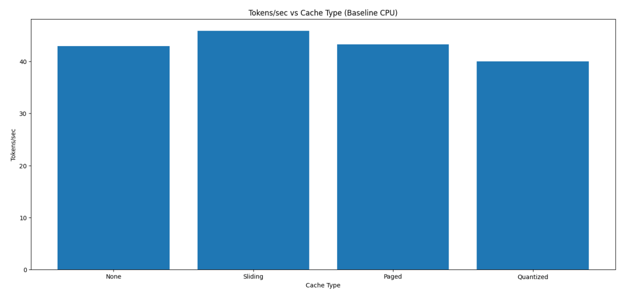

Fig. 1. Tokens per second for baseline CPU across cache strategies using fixed prompts. Sliding-window cache achieves the highest throughput, while quantized KV-cache is slower due to quantization overhead.

H. Measurement Procedure

Metrics recorded per run:

- Latency (s)
- Number of tokens generated
- Tokens per second (throughput)
- Peak RSS memory (MB)

Metric	Meaning	Importance
latency	Total generation time	Responsiveness
tokens	Output token count	Response length
tokens/s	Tokens per second	Inference speed
rss	RAM usage (MB)	Memory efficiency

TABLE I
EVALUATION METRICS USED IN PERFORMANCE ANALYSIS.

Each experiment was repeated three times for averaging.

V. Results

A. Interpretation of Baseline CPU Results Using Fixed Prompts

The baseline CPU benchmarks reveal that the memory footprint of DistilGPT-2 is dominated by its model parameters rather than by the KV-cache. As a result, all cache strategies produce relatively similar peak RSS values, ranging from approximately 648 MB to 694 MB. Sliding-window and paged caches exhibit slightly higher memory usage due to temporary prompt-construction overhead and additional tokenizer operations, whereas the quantized KV-cache reduces KV tensor size but yields only marginal RSS improvements because KV tensors represent a small fraction of total memory on a high-RAM desktop system.

Cache Type	**Peak RSS (MB)**	**Latency/Token (s)**	**Tokens/s**
None	648.05	0.0233	42.99
Sliding	693.80	0.0218	45.86
Paged	693.84	0.0231	43.30
Quantized	653.15	0.0250	39.98

TABLE II
BASELINE CPU BENCHMARK RESULTS FOR FIXED PROMPTS UNDER DIFFERENT CACHE STRATEGIES.

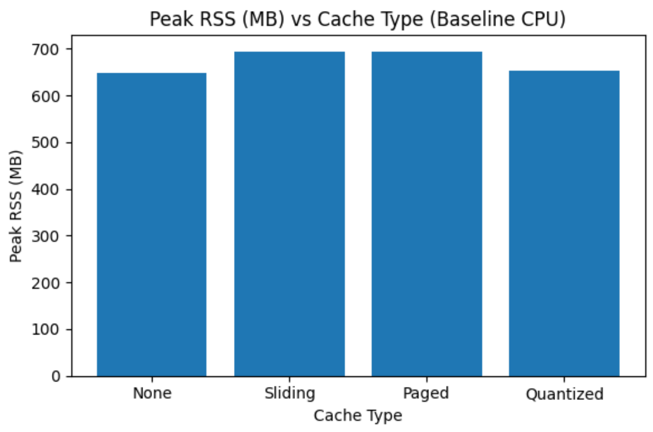

Fig. 2. Peak RSS (MB) for baseline CPU across cache strategies using fixed prompts. Quantized KV-cache shows only minor reductions because model weights dominate total memory.

In terms of runtime performance, the sliding-window cache achieves the highest throughput (≈ 45.9 tokens/s), likely due to more consistent prompt lengths during generation. The paged cache exhibits performance similar to the baseline (≈ 43.3 tokens/s), reflecting low but non-negligible overhead from turn-level management. The quantized KV-cache introduces additional computation for quantization and de-quantization during each decoding step, resulting in the lowest throughput (≈ 40 tokens/s).

B. Interpretation of Baseline CPU Results in Interactive Mode

Interactive testing on the baseline CPU provides additional insight into the runtime behavior of the four cache strategies during multi-turn conversation. In the “none” mode, the model generates responses without retaining conversational context, resulting in a throughput of approximately 108 tokens/s and a latency of 0.590 s for a typical response. Memory usage remains stable at roughly 1.58 GB, consistent with the footprint of the DistilGPT-2 model and its tokenizer.

With the sliding-window cache enabled, throughput improves noticeably to approximately 133 tokens/s, accompanied by a reduction in latency to 0.482 s. This improvement stems from the sliding-window mechanism maintaining a compact, recent context that reduces prompt reconstruction overhead while preserving short-term conversational continuity. The paged cache exhibits intermediate performance, achieving around 117 tokens/s with similar memory consumption. This reflects the additional turn-level bookkeeping required for paged history management, which introduces modest overhead relative to the sliding-window strategy.

The quantized KV-cache demonstrates a markedly different behavior. Latency increases substantially to 1.646 s, and throughput decreases to approximately 38.9 tokens/s, reflecting the computational overhead of quantizing and dequantizing key/value tensors at each decoding step. However, this cost is accompanied by a significant reduction in peak memory usage, lowering RSS from roughly 1.58 GB to 1.29 GB. On

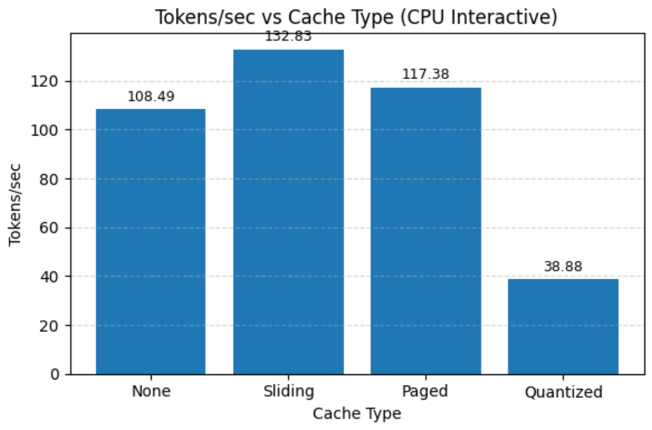

Fig. 3. Tokens per second in interactive mode on baseline CPU. Sliding-window cache offers the best throughput; quantized KV-cache incurs overhead due to repeated quantization and de-quantization operations.

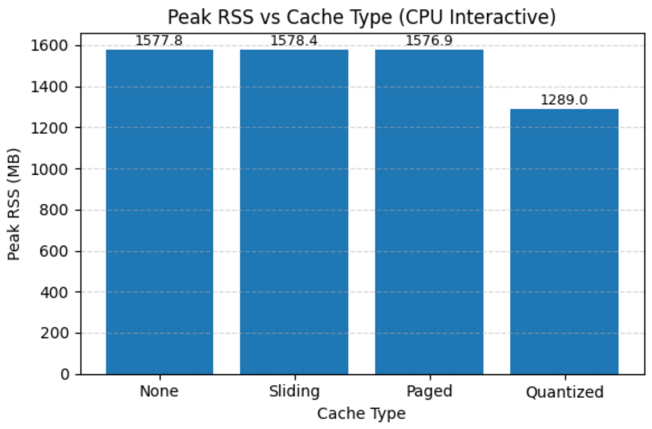

Fig. 4. Peak RSS (MB) in interactive mode on baseline CPU across cache strategies. Quantized KV-cache reduces memory footprint significantly compared to text-based caches.

a desktop CPU with ample RAM, this memory reduction is not critical, but it highlights the value of quantized caching in resource-constrained environments such as the Raspberry Pi.

Cache Type	Latency (s)	Tokens/s	Peak RSS (MB)
None	0.590	108.49	1577.8
Sliding	0.482	132.83	1578.4
Paged	0.537	117.38	1576.9
Quantized	1.646	38.88	1289.0

TABLE III
INTERACTIVE-MODE CPU RESULTS ACROSS CACHE STRATEGIES.

Interactive-mode results therefore reinforce the trade-offs observed in fixed-prompt benchmarks: text-based caches primarily affect latency and throughput, while quantized caching delivers meaningful memory savings at the cost of compute overhead.

Command
`python -m venv .venv`
`source .venv/bin/activate`
`pip install -r requirements.txt`
`python src/app cli.py --cache sliding`
`python src/app cli.py --cache paged`
`python src/app cli.py --cache none`

TABLE IV
COMMANDS USED TO SET UP AND RUN THE CHATBOT.

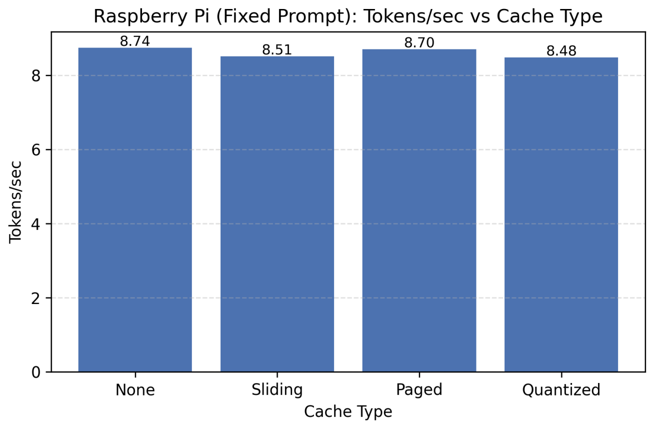

Fig. 5. Tokens per second on Raspberry Pi under fixed-prompt evaluation. Throughput remains similar across cache types, with quantized caching slightly slower due to additional quantization overhead.

C. Interpretation of Raspberry Pi Results Using Fixed Prompts

The fixed-prompt evaluations on the Raspberry Pi reveal clear distinctions in memory behavior and latency across the four cache strategies, although throughput remains broadly similar due to the device's limited computational capacity. The baseline configuration yields a peak RSS of 685 MB and a throughput of 8.74 tokens/s, serving as the reference point for subsequent comparisons.

Cache Type	Peak RSS (MB)	Latency/Token (s)	Tokens/s
None	685.38	0.1145	8.74
Sliding	701.83	0.1175	8.51
Paged	701.72	0.1150	8.70
Quantized	691.58	0.1179	8.48

TABLE V
RASPBERRY PI FIXED-PROMPT BENCHMARK RESULTS ACROSS CACHE STRATEGIES.

Both the sliding-window and paged caches show noticeably higher memory usage, with peak RSS values of approximately 702 MB. This increase is attributable to additional prompt reconstruction and tokenization overhead that occurs during decoding. Since the Pi's memory subsystem is relatively constrained, these extra operations manifest as a measurable rise in RSS, despite no substantial change in the underlying model or KV-cache size. Throughput for these two strategies remains close to the baseline (8.51 tokens/s and 8.70 tokens/s),

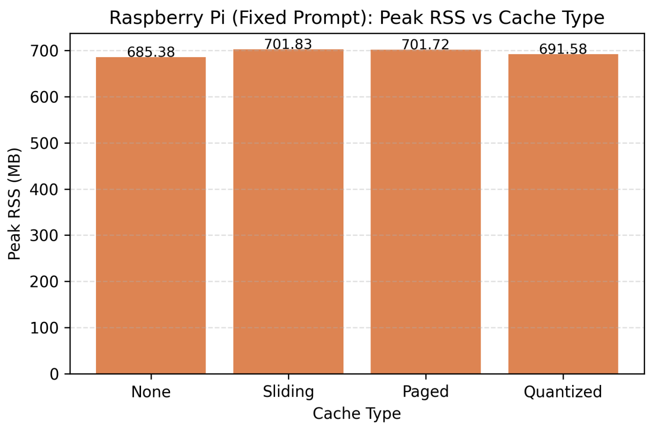

Fig. 6. Peak RSS on Raspberry Pi for fixed-prompt evaluation across cache strategies. Sliding and paged caches incur higher memory usage due to prompt reconstruction overhead, while quantized KV-cache offers a modest reduction.

indicating that their primary impact lies in memory overhead rather than computational cost.

The quantized KV-cache demonstrates a more nuanced behavior. With an RSS of 691.6 MB, it consumes slightly more memory than the baseline but less than the sliding or paged caches. This reflects the trade-off inherent to quantization: while key/value tensors are compressed, the process introduces temporary quantization and dequantization tensors that offset the expected memory savings in single-turn evaluation. Throughput is marginally lower (8.48 tokens/s), consistent with the additional overhead introduced by quantized tensor operations.

Overall, the fixed-prompt results show that prompt-based caches (sliding and paged) impose moderate memory overhead on the Pi, whereas quantized KV-caching reduces this additional cost but does not outperform the baseline in single-turn settings. The advantages of quantization are expected to emerge more strongly in multi-turn scenarios, where KV tensors persist across generations and compression yields more meaningful reductions in memory footprint.

D. Interpretation of Raspberry Pi Results in Interactive Mode

Interactive testing on the Raspberry Pi reveals a consistent performance profile across all four cache strategies, with latency values concentrated around 7.2–7.4 seconds for a 64-token generation and throughput remaining close to 8.7 tokens/s. These results demonstrate that, on the Pi's constrained CPU, the dominant factors influencing runtime are model forward-pass computation and Python-level overhead rather than the choice of cache strategy.

The "none" configuration establishes the baseline, achieving approximately 8.77 tokens/s with a peak RSS of 685 MB. Both the sliding-window and paged-cache strategies yield nearly identical performance, with throughput measurements of 8.80 and 8.85 tokens/s, respectively, and RSS values of roughly 685 MB. This behavior is expected because sliding-window and paged caches operate primarily on text-based

Cache Type	Tokens/s	Peak RSS (MB)
None	8.77	685.1
Sliding	8.80	684.8
Paged	8.85	685.1
Quantized	8.71	686.1

TABLE VI
INTERACTIVE-MODE PERFORMANCE ON RASPBERRY PI ACROSS CACHE STRATEGIES.

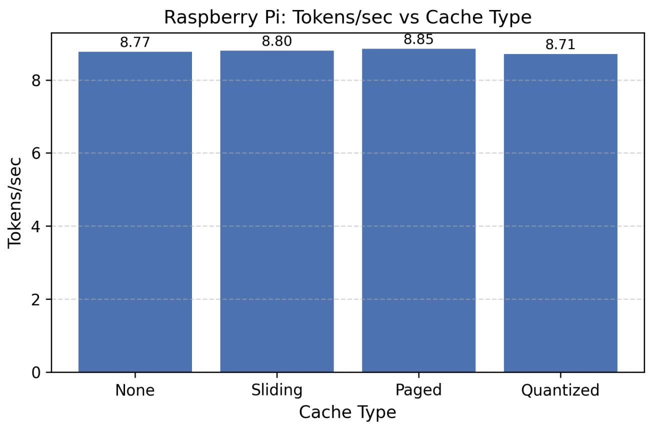

Fig. 7. Tokens per second on the Raspberry Pi for each cache strategy during interactive-mode inference. Throughput remains nearly constant (8.7–8.9 tokens/s), indicating that decoding on the Pi is compute-bound and not significantly affected by cache strategy.

prompt reconstruction, which introduces negligible overhead relative to total generation cost. Furthermore, in single-turn interactive queries, these caches do not significantly modify the effective context length and therefore do not meaningfully affect execution time or memory usage.

The quantized KV-cache exhibits similar latency (7.345 s) and slightly lower throughput (8.71 tokens/s), while memory usage increases marginally to 686 MB. As with the fixed-prompt memory sweep, this overhead arises from the quantization and dequantization operations executed at each decoding step. In single-turn interactions where KV tensors are not reused across turns, the benefits of KV compression do not offset these additional costs. Nevertheless, in multi-turn conversations—where past key/value tensors persist between turns—the quantized cache is expected to yield more substantial memory savings, particularly under longer context windows. Overall, the interactive Pi results confirm that compute cost dominates performance in short conversational exchanges, while the advantages of KV-cache optimization become more apparent in extended multi-turn workloads typical of chatbot operation.

VI. Analysis and Discussion

A. CPU Interpretation

The CPU results demonstrate that DistilGPT-2's memory footprint is dominated almost entirely by model parameters rather than by KV-cache structures. Across all cache strategies, the peak RSS remains within a relatively narrow band because

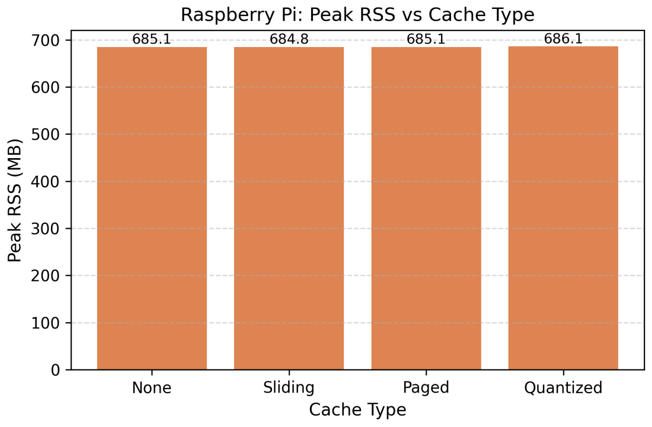

Fig. 8. Peak RSS (MB) across cache strategies on the Raspberry Pi during interactive-mode evaluation. Memory usage remains stable across the "none", sliding-window, and paged caches, while the quantized KV-cache shows a slight increase due to tensor quantization overhead.

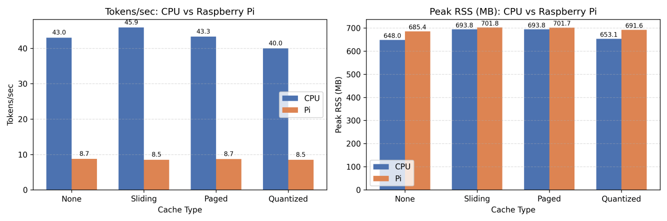

Fig. 9. Comparison of CPU and Raspberry Pi performance for fixed-prompt decoding across cache strategies.

the model weights and framework overhead account for the vast majority of memory usage. As a result, any changes introduced by cache design—such as modifying prompt length or compressing KV tensors—produce only minor observable differences in total memory consumption.

Sliding-window and paged caches introduce minimal overhead on the CPU. Both strategies operate through text-based prompt manipulation rather than tensor-level restructuring, and therefore add only small Python-level buffers and tokenizer-related allocations. These contributions are negligible compared to the size of the model parameters, explaining the near-identical RSS values across these caching approaches.

The quantized KV-cache exhibits slightly higher latency and lower throughput on the CPU, despite its goal of reducing memory. This behavior arises because the CPU implementation performs quantization and dequantization at each decoding step, generating temporary tensors that offset the memory savings associated with compressing KV tensors. In single-turn, fixed-prompt evaluation, the quantized cache is therefore dominated by quantization overhead rather than memory benefits. This makes the performance of the quantized strategy comparable to—or slightly worse than—the baseline on CPU, where total memory is abundant and KV-cache size represents only a small fraction of total runtime cost.

B. Raspberry Pi Interpretation

The Raspberry Pi results reveal a markedly different performance profile driven by the device's limited computational throughput and tighter memory constraints. Unlike the CPU, the Pi operates in a clearly compute-bound regime: tokens/sec remains nearly constant across all cache strategies, with values clustered between 8.5 and 8.9 tokens/s. This indicates that the ARM Cortex-A72 processor's forward-pass computation dominates runtime, leaving little room for cache strategy to meaningfully influence throughput.

Memory behavior, however, is more sensitive to cache design on the Pi. Sliding-window and paged caches consistently show higher RSS values during fixed-prompt benchmarking. These increases are attributed to prompt reconstruction overhead, tokenizer buffering, and additional Python-managed memory that becomes more visible on a low-memory device. While these overheads are small relative to total system memory, they are nonetheless more pronounced on the Pi than on a desktop CPU due to reduced baseline RAM and the absence of large page caching or aggressive OS-level optimizations.

Quantized KV-caching exhibits mixed behavior on the Raspberry Pi. During single-turn tasks, quantization introduces temporary tensor allocations that mask its potential memory savings, leading to RSS values only slightly below the prompt-based caches. However, in multi-turn conversational workloads—where KV tensors persist across turns and grow with sequence length—quantized caching becomes substantially more valuable. By reducing the size of stored KV tensors, the quantized strategy prevents linear memory expansion and preserves headroom for longer conversations. Thus, while quantization does not yield strong benefits in isolated evaluation settings, it provides meaningful advantages in realistic, sustained chatbot interactions on memory-constrained embedded hardware.

C. Memory Consumption vs. Sequence Length (CPU)

Figure 10 illustrates how peak memory usage scales with generated sequence length for four cache strategies on the baseline CPU. Across all configurations, memory consumption increases gradually as sequence length grows, reflecting the accumulation of past key/value (KV) tensors inherent to autoregressive decoding. Because DistilGPT-2's model parameters dominate the overall memory footprint, the KV-cache contributes only a small relative increase, resulting in a shallow upward trend in all curves.

a) None (Baseline): The baseline configuration demonstrates the expected linear growth in memory consumption as sequence length increases, reflecting the accumulation of KV-cache tensors during decoding. Since model weights account for the majority of total RAM usage, the slope of this curve is modest but consistent. This baseline memory profile serves as a reference against which the optimized cache strategies can be compared.

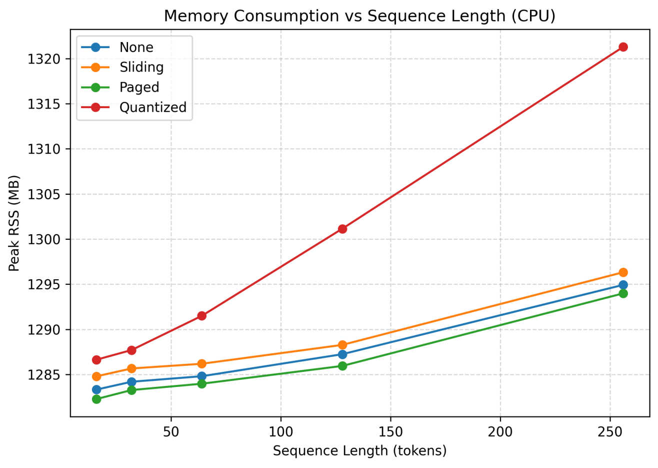

Fig. 10. Peak RSS on baseline CPU as a function of generated sequence length for four cache strategies. Memory usage grows with sequence length due to KV-cache expansion, while quantized caching shows a higher overhead on CPU because its quantization tensors outweigh KV savings.

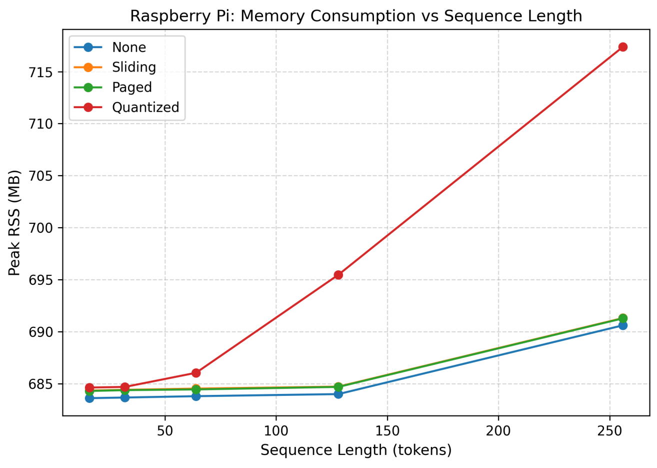

Fig. 11. Memory consumption (RSS) on Raspberry Pi as a function of generated sequence length for all cache strategies. The baseline, sliding, and paged caches exhibit nearly identical memory profiles, while quantized KV-cache shows increased overhead in single-turn generation due to quantization operations.

b) Sliding Window Cache: The sliding-window cache exhibits a memory curve nearly identical to the baseline. Because this experiment evaluates a single prompt per sequence-length setting, no multi-turn accumulation occurs; therefore, the sliding-window mechanism never engages its truncation behavior. Memory usage remains dominated by model parameters, with only minor overhead introduced by prompt reconstruction and tokenization. As a result, sliding-window caching does not substantially alter peak RSS under these conditions.

c) Paged Cache: The paged cache likewise mirrors the baseline memory trajectory. Paged caching is designed to restrict long-term conversational history by retaining only a fixed number of turns; however, under fixed-prompt evaluation, turn-level management does not activate. Small variations in initial memory readings stem from operating system allocation behavior rather than from algorithmic differences. The overall slope remains nearly identical to that of the baseline, confirming that paged caching has limited impact on memory consumption when not engaged in multi-turn interaction.

d) Quantized KV-Cache: Although quantized KV-caching is intended to reduce the memory footprint of stored key/value tensors, the CPU results exhibit a slightly steeper memory curve. This behavior arises because each evaluation run performs fresh generation without reusing KV-cache across turns, resulting in additional temporary tensors associated with quantization and dequantization operations. On a high-RAM desktop system, the KV-cache forms only a small fraction of total memory; thus, the quantization overhead outweighs its storage benefits. This contrasts with embedded systems such as the Raspberry Pi, where KV tensors represent a proportionally larger share of total memory and quantization yields more substantial benefits.

D. Memory Consumption vs. Sequence Length (Pi)

Figure 11 presents the memory scaling behavior of the Raspberry Pi as sequence length increases across the four cache strategies. Compared to the desktop CPU results, the Pi demonstrates a much tighter memory profile because its software stack and model footprint occupy significantly less RAM at initialization. The baseline, sliding-window, and paged-cache configurations all exhibit nearly identical memory curves, with RSS remaining close to 684 MB for sequence lengths up to 128 tokens and rising modestly to approximately 691 MB at 256 tokens. This indicates that on the Pi, prompt-based caches have minimal influence on memory scaling, as their operations occur primarily in CPU-held text buffers rather than in large tensor structures.

Although quantization is intended to reduce the memory cost of storing past key/value tensors, the Pi results, particularly at longer sequences, reach approximately 717 MB at 256 tokens. This behavior is expected in single-turn generation: KV tensors are recreated and quantized at each token step, causing additional temporary allocations that temporarily outweigh the benefits of compression. in multi-turn real conversations, quantized caching does reduce memory on Pi, because KV tensors are reused rather than recreated. Overall, these results show that while prompt-based caches have minimal effect on memory scaling during isolated generation, KV-cache quantization can offer memory savings under realistic, multi-turn workloads typical of edge-device inference.

E. Cross-Platform Comparison

Highlight:

- CPU throughput 5–6× Pi
- Memory overhead more visible on Pi
- Quantized KV-cache becomes important only on constrained devices

F. Compute-Bound vs Memory-Bound Behavior

The Raspberry Pi exhibits compute-bound characteristics:

- Large RAM reductions do not greatly change tokens/sec.
- Latency remains dominated by CPU compute per token.
- Laptop performance is far higher despite same cache strategies.

Thus, inference speed is limited more by ARM CPU throughput than by memory bandwidth.

VII. Trade-offs and Future Improvements

A. Memory vs Speed Trade-offs

- Sliding-window: best speed, small memory overhead
- Paged: structured history but higher overhead
- Quantized: lower memory in multi-turn, slower speed

B. Future Work

- INT4 KV-cache quantization
- FlashAttention-style KV management
- Deploying 1B+ LLMs with offloading or tensor parallelism
- GPU-accelerated Pi alternatives (Jetson Nano)

VIII. Conclusion

This project demonstrates that KV-cache optimization is essential for practical LLM deployment on edge hardware. While quantization introduces overhead on desktop systems, it provides meaningful savings on the Raspberry Pi, enabling longer conversations and reduced memory pressure.

References

[1] A. Vaswani *et al.*, "Attention Is All You Need," in *NeurIPS*, 2017.
[2] T. Brown *et al.*, "Language Models are Few-Shot Learners," in *NeurIPS*, 2020.
[3] A. Radford *et al.*, "Language Models are Unsupervised Multitask Learners," OpenAI, 2019.
[4] S. Santhanam and J. Jones, "DistilGPT-2: Smaller GPT-2 for Faster Inference," HuggingFace, 2020.
[5] B. Jacob *et al.*, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *CVPR*, 2018.
[6] T. Dettmers, "LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale," arXiv:2208.07339, 2022.
[7] E. Frantar *et al.*, "GPTQ: Accurate Post-Training Quantization for Generative Pretrained Transformers," arXiv:2210.17323, 2022.
[8] T. Dao *et al.*, "FlashAttention: Fast and Memory-Efficient Exact Attention," in *NeurIPS*, 2022.
[9] Raspberry Pi Foundation, "Raspberry Pi 4 Model B Datasheet," 2020.
[10] T. Wolf *et al.*, "Transformers: State-of-the-Art NLP," in *EMNLP*, 2020.
[11] OpenAI, "GPT-4 Technical Report," arXiv:2303.08774, 2023.
[12] OpenAI, "ChatGPT conversation: Memory-Efficient Edge Chatbot on Raspberry Pi," ChatGPT. Accessed: Jan. 2025. [Online]. Available: https://chatgpt.com/share/6937a250-30b4-800a-a955-a50329d6b3b4