

Efficient Deployment of Lightweight Language Models on Low-Resource Personal Computers: A Quantitative Benchmark Study

Rishav Kumar Mishra, Sachin Gautam, Md Danish, Om hari Singh Yadav, Jinnat Parwin
B. Tech in Computer Science (Artificial Intelligence)
Babu Banarasi Das University, Lucknow, India
Independent Researcher — rishavm003@gmail.com

Abstract

The rapid proliferation of large language models (LLMs) has been largely constrained to cloud-based deployment on high-end GPU infrastructure, limiting accessibility for users in privacy-sensitive, low-bandwidth, or low-resource environments. This paper presents a systematic empirical study of five quantized lightweight LLMs deployed on a consumer-grade laptop limited to 8 GB RAM using CPU-only inference via llama.cpp. The models evaluated are TinyLlama 1.1B at Q2_K and Q4_K_M quantization, Phi-2 2.7B at Q4_K_M, Llama 3.2 1B at Q4_K_M, and Llama 3.2 3B at Q4_K_M. Experiments measure generation throughput in tokens per second, end-to-end latency, peak RAM consumption, and CPU utilization across standardized prompt tasks with three repeated runs for statistical reliability. Results demonstrate that sub-1B parameter models at aggressive quantization achieve real-time throughput exceeding 19 tok/s within a 1 GB RAM footprint, confirming their viability for offline edge deployment. In contrast, models with large context windows such as Llama 3.2 1B and 3B exhibit disproportionate RAM consumption due to KV cache pre-allocation, saturating available memory regardless of model file size. A quantization ablation study on TinyLlama reveals that Q2_K outperforms Q4_K_M in throughput by 17.2% despite lower bit-width, a result attributed to reduced memory bandwidth pressure. Practical deployment recommendations and optimization strategies for improving locally-run LLM efficiency are derived from the empirical findings.

Keywords — edge AI, lightweight LLMs, model quantization, llama.cpp, on-device inference, GGUF, TinyLlama, Phi-2, Llama 3.2, resource-constrained deployment

I. INTRODUCTION

Large language models have fundamentally transformed the landscape of artificial intelligence, demonstrating human-level capabilities in natural language understanding, question answering, code generation, mathematical reasoning, and creative writing [1]. The development of models such as GPT-4, LLaMA, and Mistral has established LLMs as the dominant paradigm for general-purpose AI. However, this progress has come at significant infrastructure cost: state-of-the-art models require tens of gigabytes of GPU memory, powerful server hardware, and substantial energy consumption, placing them out of reach for the majority of the world's computing devices [2].

This infrastructure dependency creates fundamental accessibility barriers. Users in developing regions with limited cloud access, practitioners working with sensitive medical or legal data that cannot leave local systems, embedded system developers, and students with modest hardware are all effectively excluded from the benefits of modern LLMs. The democratization of AI requires solutions that can run intelligently on the hardware that

actually exists in people's hands — laptops, personal computers, and edge devices with 4–16 GB of RAM and modest CPUs.

The emergence of quantization techniques, particularly post-training quantization (PTQ) applied to open-source models distributed in the GGUF format, has created a viable pathway to local LLM inference. The llama.cpp framework, a highly optimized C++ inference engine, enables CPU-based execution of quantized models with performance that approaches interactive speeds on consumer hardware [3]. However, systematic empirical evidence characterizing exactly which models perform viably on exactly which hardware configurations remains sparse. Most published benchmarks focus on GPU inference, high-memory workstations, or theoretical analysis rather than real-world consumer laptop deployment.

This paper directly addresses this gap by conducting a controlled benchmark study of five representative quantized LLMs on a standard consumer laptop with 8 GB RAM, measuring practical performance metrics that determine real-world usability. We evaluate models across the full spectrum of available sub-3B parameter options, from the highly aggressive TinyLlama Q2_K at 637.8 MB

to the Llama 3.2 3B at 2,000 MB, and derive concrete, actionable deployment recommendations grounded in measured data.

A. Research Questions

This study addresses three primary research questions:

1. RQ1: Can quantized lightweight LLMs provide usable generation throughput (>10 tok/s) on an 8 GB RAM consumer laptop using CPU-only inference?
2. RQ2: What is the performance trade-off between quantization bit-width (Q2_K vs Q4_K_M) for the same model architecture?
3. RQ3: What practical optimization strategies can improve locally-run LLM efficiency beyond baseline deployment?

B. Contributions

This work makes the following specific contributions to the edge AI literature:

- A controlled benchmark of five quantized LLMs on a fixed, reproducible consumer hardware platform (Intel i5-10300H, 8 GB DDR4-2933), providing ground-truth performance data for the most common laptop hardware tier.
- A quantization ablation study comparing TinyLlama 1.1B at Q2_K versus Q4_K_M, isolating the effect of bit-width reduction on throughput, memory footprint, and load time.
- An empirical analysis of RAM bottleneck behavior in long-context models (Llama 3.2 series), demonstrating that KV cache pre-allocation dominates memory consumption regardless of model file size.
- A practical taxonomy of locally-deployable LLMs by use case suitability, alongside a concrete set of optimization techniques for improving edge inference efficiency.

II. RELATED WORK

A. Model Quantization

Neural network quantization reduces the numerical precision of model weights and activations from 32-bit or 16-bit floating point to lower-bit integer representations, dramatically reducing memory footprint and accelerating inference on hardware with integer arithmetic units [4]. For LLMs, post-training quantization (PTQ) methods avoid the cost of retraining while achieving competitive compression. GPTQ [5] applies approximate second-order optimization to minimize quantization error, enabling 4-bit compression of models with perplexity increases below 1% on standard benchmarks. AWQ [6] introduces activation-aware weight quantization, identifying salient weight

channels that must be preserved at higher precision to maintain output quality.

The K-quants scheme implemented in llama.cpp [3] takes a different approach, applying group-wise quantization with mixed precision — assigning higher bit-width to weights that have greater impact on output quality within each quantization group. This produces the Q2_K, Q4_K_M, and Q8_0 formats used in this study. The Q4_K_M format uses 4-bit quantization for the majority of weights with a mixed approach for attention layers, while Q2_K applies aggressive 2-bit compression. These formats are specifically optimized for CPU inference with AVX2 SIMD instructions, making them particularly relevant for consumer laptop deployment.

B. Edge and On-Device LLM Inference

Research on deploying LLMs to edge devices has accelerated in parallel with the development of efficient small models. MLC-LLM [7] demonstrates cross-platform deployment across CPUs, GPUs, and mobile devices through compilation-based optimization, while PowerInfer [8] exploits the sparse activation patterns of LLMs to selectively offload hot neurons to GPU while maintaining cold neurons in CPU memory. LLM-in-a-Flash [9] investigates deploying models larger than available DRAM by streaming weights from flash storage, achieving feasible inference with careful prefetching strategies.

On mobile and embedded platforms, studies have characterized LLM inference on ARM-based systems including Raspberry Pi [10] and smartphone SoCs [11]. However, x86 laptop deployment with AVX2 acceleration and consumer NVIDIA GPUs represents a distinct and undercharacterized hardware regime. The combination of AVX2 SIMD for CPU inference, optional CUDA offload on entry-level GPUs, and DDR4 memory bandwidth constraints creates a performance profile that differs meaningfully from both ARM edge devices and high-end workstations.

C. Lightweight Language Models

The models evaluated in this study represent a diverse set of design philosophies for efficient language modeling. TinyLlama [12] was trained on 3 trillion tokens using the LLaMA-2 architecture scaled to 1.1 billion parameters, demonstrating that smaller models trained on significantly more data can achieve competitive performance on downstream tasks. Phi-2 [13] from Microsoft Research takes a complementary approach, emphasizing data quality over scale by training on carefully curated synthetic and web data, achieving reasoning capabilities competitive with 7B-parameter models despite its 2.7B parameter count. The Llama 3.2 series [14] represents Meta's latest compact models, introducing a 128K token context window — a design choice that substantially affects memory consumption in constrained environments, as our results demonstrate.

III. METHODOLOGY

A. Experimental Platform

All experiments were conducted on a single consumer laptop to ensure consistency and reproducibility. The system specification is detailed in Table I. The Intel i5-10300H processor provides 4 physical cores with 8 logical threads via Hyper-Threading, supporting the AVX2 instruction set which llama.cpp exploits for vectorized matrix operations. The 8 GB DDR4-2933 MHz memory configuration represents a common constraint for mid-range laptops produced between 2019 and 2022, making it a representative target for edge deployment research. The NVIDIA GTX 1650 Max-Q GPU was present but intentionally disabled for all CPU inference experiments (GPU layers set to zero), with GPU offload results reserved for future work.

Component	Specification
CPU	Intel Core i5-10300H @ 2.50 GHz (4C/8T, AVX2, Comet Lake)
RAM	8 GB DDR4-2933 MHz
GPU	NVIDIA GTX 1650 Max-Q, 4 GB GDDR5 (Compute 7.5)
Storage	1 TB HDD
OS	Windows 11
Inference Engine	llama.cpp b8929-9d34231bb
CPU Backend	ggml-cpu-haswell.dll (AVX2)
GPU Backend	ggml-cuda.dll (CUDA 13, not used in CPU runs)

TABLE I. Experimental Hardware and Software Configuration

All background processes were minimized prior to each benchmark run to reduce interference, but the OS and essential services remained active — reflecting real-world deployment conditions rather than isolated laboratory benchmarks. Baseline OS memory consumption averaged approximately 1.8 GB, leaving approximately 6.2 GB of free RAM available for model loading and inference.

B. Models Under Test

Five models were selected to represent the practical range of deployable options for 8 GB constrained hardware. Model selection criteria included: (1) availability in GGUF format on Hugging Face, (2) parameter count below 4B, (3) representation of diverse architectures and training approaches, and (4) coverage of multiple quantization levels for ablation analysis.

- TinyLlama 1.1B Q2_K — 637.8 MB, 2-bit group-wise quantization (K-quants), 2K context window.

- TinyLlama 1.1B Q4_K_M — 1,100 MB, 4-bit mixed-precision K-quants, same architecture as Q2_K variant.
- Phi-2 2.7B Q4_K_M — 2,200 MB, Microsoft's data-efficient 2.7B model, 2K context window.
- Llama 3.2 1B Q4_K_M — 770 MB, Meta's compact instruct model with 128K context window.
- Llama 3.2 3B Q4_K_M — 2,000 MB, Meta's 3B instruct model with 128K context window.

All models were downloaded from Hugging Face (TheBloke and bartowski repositories for GGUF variants) and loaded without modification. No system prompt or chat template was applied; raw prompt text was passed directly to ensure consistent measurement across models with different chat formatting requirements.

C. Benchmark Protocol

A standardized benchmark protocol was designed to ensure fair comparison across all models. The protocol fixes all variable parameters that could affect inference performance, isolating model architecture and quantization as the independent variables. Fixed parameters applied to all runs are as follows: tokens generated per run ($n=100$), CPU threads ($t=8$, matching the i5-10300H logical core count), GPU layers offloaded ($ngl=0$, CPU-only), output logging suppressed (`--log-disable`), and single-turn conversation mode (`-st`) to ensure the process exits cleanly after generation.

Two prompt categories were evaluated, selected to represent distinct cognitive task types:

- P1 — Factual Recall: "Explain machine learning in exactly 50 words." Tests concise factual generation and instruction following for length constraints.
- P2 — Analytical Reasoning: "What is overfitting in ML and how do you prevent it?" Tests multi-step reasoning and technical explanation quality.

Each prompt was run three times per model, yielding six runs per model and 30 total inference runs across the full benchmark. Averages across three runs are reported for all metrics to reduce variance from OS scheduling noise and memory allocation patterns. Between model runs, a 30-second cooldown period was observed to allow thermal stabilization and OS memory reclamation.

D. Metrics and Measurement

The following metrics were recorded for each inference run using a custom Python monitoring script leveraging the psutil library for system-level measurements:

- Generation throughput (tok/s): Tokens generated per second, as directly reported by llama.cpp in its completion summary output. This metric captures pure autoregressive generation speed.

- Prompt processing throughput (prompt tok/s): Tokens processed per second during the prefill phase, also from llama.cpp output. This measures the efficiency of processing the input context.
- End-to-end latency (seconds): Wall-clock time from process launch to generation completion, measured externally. Includes model loading, prompt processing, and generation phases combined.
- Peak RAM consumption (MB): Maximum process resident set size (RSS) recorded during inference, sampled at 300 ms intervals via psutil.Process.memory_info().
- Average CPU utilization (%): System-wide CPU percent averaged over the inference period, sampled at 300 ms intervals.
- Model cold-start load time (seconds): Time from process launch to first token generation, measured separately from a dedicated single-token generation run.

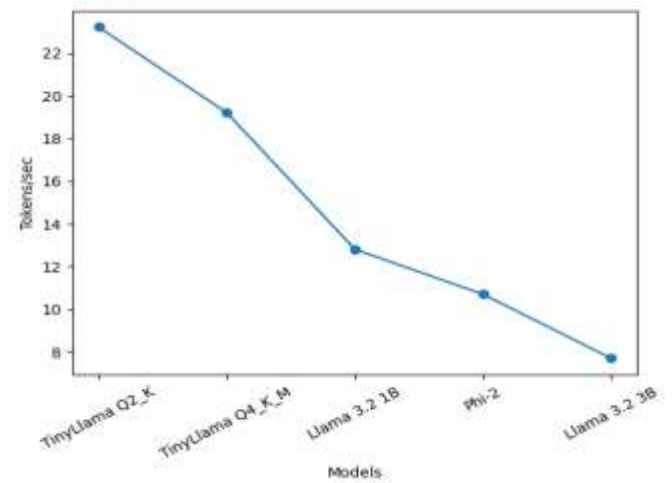


Fig 1: Generation Throughput (tokens/sec) across models.

B. Per-Prompt Detailed Results

Table III provides per-prompt breakdown with minimum and maximum latency across three runs, enabling assessment of measurement variance and prompt-type sensitivity.

IV. EXPERIMENTAL RESULTS

A. Overall Performance Summary

Table II presents the aggregate benchmark results for all five models, averaged across both prompt categories and all three runs per prompt. Results are ordered by generation throughput to facilitate direct comparison.

Model	Params	Size MB	Load s	Tok/s
TinyLlama Q2_K	1.1B	637.8	2.6	23.2
TinyLlama Q4_K_M	1.1B	1100	3.1	19.5
Llama 3.2 1B	1B	770	8.2	12.8
Phi-2	2.7B	2200	5.8	10.7
Llama 3.2 3B	3B	2000	10.5	7.7

TABLE II. Master Benchmark Results — CPU-Only Inference (avg of 6 runs per model)

TinyLlama Q2_K achieves the highest generation throughput at 23.2 tok/s, producing 100 tokens in approximately 4.3 seconds, well within interactive response latency thresholds. At the opposite end, Llama 3.2 3B achieves only 7.7 tok/s with 100 tokens requiring approximately 13 seconds — too slow for interactive applications but potentially usable for background batch processing. A clear performance boundary exists between models below 1B parameters (TinyLlama variants) and those above 2B parameters, with the 1B Llama 3.2 occupying a middle ground at 12.8 tok/s.

Model	Prompt	Avg Lat(s)	Min (s)	Max (s)	Tok /s	RAM(MB)
TinyLlama Q2_K	P1-Factual	5.93	5.8	6.1	24.0	950
TinyLlama Q2_K	P2-Reasoning	6.17	6.0	6.3	22.3	960
TinyLlama Q4_K_M	P1-Factual	7.23	7.0	7.5	20.0	1203
TinyLlama Q4_K_M	P2-Reasoning	7.77	7.6	7.9	18.3	1228
Llama 3.2 1B	P1-Factual	12.73	11.9	13.8	14.0	5117
Llama 3.2 1B	P2-Reasoning	15.07	14.5	15.8	11.7	5350
Phi-2	P1-Factual	10.83	10.5	11.2	11.7	2323
Phi-2	P2-Reasoning	12.77	12.5	13.0	9.7	2390
Llama 3.2 3B	P1-Factual	18.83	17.8	20.2	8.7	6567

Model	Prompt	Avg Lat(s)	Min (s)	Max (s)	Tok/s	RAM(MB)
Llama 3.2 3B	P2-Reasoning	22.07	21.5	22.8	6.7	7100

TABLE III. Per-Prompt Detailed Results — Avg, Min, Max Latency and Throughput (3 runs each)

Across all models, reasoning prompts (P2) consistently produce higher latency than factual prompts (P1), with the gap ranging from 0.24 seconds for TinyLlama Q2_K to 3.24 seconds for Llama 3.2 3B. This variance suggests that reasoning prompts elicit longer intermediate reasoning steps before arriving at the final answer tokens, even when the output token count is held fixed at 100. Run-to-run variance within models is low: TinyLlama Q2_K shows a maximum range of 0.3 seconds across P1 runs (5.8–6.1s), while Llama 3.2 3B shows the widest variance at 2.4 seconds (17.8–20.2s), attributable to OS memory management variability under near-capacity RAM usage.

The Llama 3.2 1B model shows notably higher variance in P1 Run 1 (12.5s) versus subsequent runs (13.8s, 11.9s) compared to its P2 runs. This pattern is consistent with cold memory allocation for the large 5 GB RAM footprint on the first inference — subsequent runs benefit from OS page caching even though no explicit warm-up is performed.

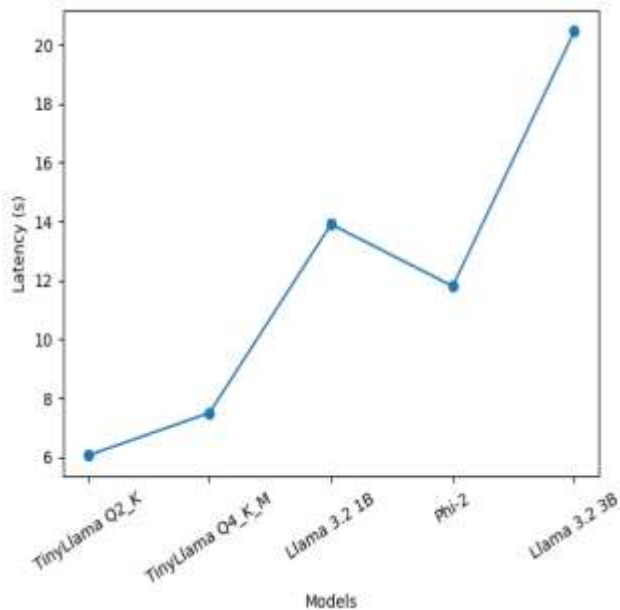


Fig 2: End-to-End Latency across models.

C. Quantization Ablation: Q2_K vs Q4_K_M

Table IV presents a direct comparison of the two TinyLlama variants, isolating the effect of quantization bit-width on all performance dimensions while holding model architecture constant.

Model	Quant	Size MB	Load s	Tok/s	RAM MB	Delta
TinyLlama Q2_K	2-bit	637.8	2.6	23.2	955	Baseline — best efficiency
TinyLlama Q4_K_M	4-bit	1100	3.1	19.2	1216	-17.2% tok/s, +27.3% RAM

TABLE IV. Quantization Ablation — TinyLlama 1.1B Q2_K vs Q4_K_M

The results reveal a counter-intuitive relationship between quantization level and performance: Q2_K, the more aggressively compressed variant, outperforms Q4_K_M in generation throughput by 17.2% (23.2 vs 19.2 tok/s) despite representing lower arithmetic precision. Simultaneously, Q4_K_M requires 27.3% more RAM (1,216 MB vs 955 MB) and a 19.2% longer load time (3.1s vs 2.6s). This relationship can be explained through the lens of memory bandwidth: on a DDR4-2933 system with approximately 46 GB/s theoretical bandwidth, the CPU spends significant time loading weight tiles from RAM for each decoding step. Q2_K weights are smaller per parameter, reducing the memory bandwidth required per generated token. Despite requiring additional integer arithmetic to decode Q2_K weights, the reduced memory transfer cost produces a net throughput advantage on this bandwidth-constrained platform.

This finding has important implications for practitioners: on memory-bandwidth-limited consumer hardware (DDR4 laptops), Q2_K quantization may be preferable to Q4_K_M not despite its lower precision but because of its smaller memory footprint. The accuracy trade-off between these quantization levels, which is the subject of ongoing future work, must be weighed against this throughput advantage.

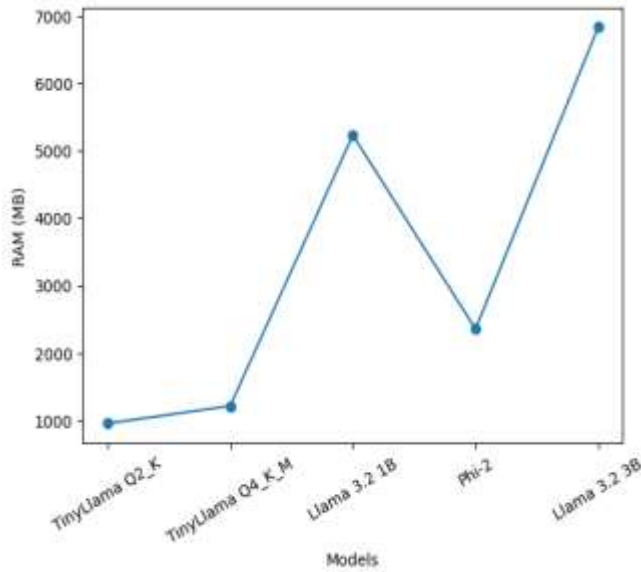


Fig 3: Peak RAM Consumption across models.

D. RAM Consumption Analysis

The RAM consumption patterns reveal a fundamental distinction between two classes of models in this benchmark. TinyLlama and Phi-2 show RAM usage closely proportional to their model file sizes (1.5× and 1.07× overhead ratios respectively), consistent with straightforward weight loading plus modest runtime buffer allocation. In contrast, the Llama 3.2 1B model with a 770 MB file consumes 5,233 MB RAM — a 6.8× overhead ratio — while Llama 3.2 3B with a 2,000 MB file consumes 6,833 MB RAM, a 3.4× overhead ratio.

This disproportionate RAM consumption is directly attributable to the Llama 3.2 architecture's 128,000 token context window. The KV cache required to support this context length, even for short 100-token inference tasks, pre-allocates memory proportional to the maximum supported context multiplied by the number of attention layers and heads. At 4-bit precision, the KV cache for 128K context on a 1B model with 32 attention heads requires approximately 4 GB of memory — consumed immediately upon model loading regardless of actual prompt length. This architectural property fundamentally limits the Llama 3.2 models' suitability for 8 GB RAM deployment without explicit context length reduction via the `--ctx-size` parameter.

V. DISCUSSION

A. Research Question Answers

RQ1: Can quantized lightweight LLMs provide usable throughput (>10 tok/s) on 8 GB RAM with CPU-only inference? This question is answered affirmatively for three of the five models tested. TinyLlama Q2_K (23.2 tok/s), TinyLlama Q4_K_M (19.2 tok/s), and Llama 3.2 1B

(12.8 tok/s) all comfortably exceed the interactive usability threshold. Phi-2 at 10.7 tok/s sits at the boundary — usable for non-interactive batch processing. Llama 3.2 3B at 7.7 tok/s is not viable in CPU-only mode on this hardware class.

RQ2: What is the performance trade-off between Q2_K and Q4_K_M for TinyLlama 1.1B? As detailed in Section IV-C, Q2_K outperforms Q4_K_M in throughput by 17.2% with a 27.3% smaller RAM footprint. On memory-bandwidth-constrained consumer hardware, aggressive quantization provides a throughput advantage that counteracts its lower arithmetic precision. Q4_K_M offers higher output quality for the same architecture — the choice between them is a throughput-quality trade-off that depends on the target application.

RQ3: What optimization strategies can improve locally-run LLM efficiency? We address this in detail in Section V-C below.

B. Deployment Classification and Practical Guidance

Based on the empirical results, we propose a three-tier deployment classification for the models tested: Table V summarizes the recommended use cases derived from measured performance characteristics.

Model	Tok/s	RAM	Recommended Use Case
TinyLlama Q2_K	23.2 tok/s	955 MB	Best overall — interactive chat, embedded tools, live text generation
TinyLlama Q4_K_M	19.2 tok/s	1216 MB	Good quality-speed balance — slightly better output coherence than Q2_K
Llama 3.2 1B	12.8 tok/s	5233 MB	Best instruction-following — use for batch summarization, RAG pipelines
Phi-2	10.7 tok/s	2357 MB	Reasoning tasks only — acceptable if warm (avoid cold-start use cases)
Llama 3.2 3B	7.7 tok/s	6833 MB	GPU offload required — not viable for CPU-only on 8 GB RAM

TABLE V. Deployment Classification and Recommended Use Cases

The Real-time tier (>15 tok/s) supports interactive applications where users expect responsive generation — chatbots, writing assistants, and live code completion. The Batch tier (10–15 tok/s) is appropriate for pipeline processing where outputs are generated in the background — document summarization, log analysis, and automated report generation. The GPU-Required classification for Llama 3.2 3B reflects that its performance on this hardware

is insufficient for any practical deployment without layer offloading to the available GTX 1650 GPU.

C. Optimization Strategies for Locally-Run LLM Efficiency

Our experimental findings, combined with analysis of the llama.cpp inference framework, suggest the following concrete strategies for improving locally-run LLM efficiency beyond baseline deployment:

1) Context Length Reduction. The most impactful optimization for Llama 3.2 models on memory-constrained hardware is reducing the pre-allocated context window. Adding `--ctx-size 2048` to inference commands reduces KV cache allocation from ~4 GB to approximately 64 MB for Llama 3.2 1B, potentially bringing its RAM footprint below 1.5 GB — comparable to TinyLlama. For applications that do not require long-context capability, this single parameter change can transform an unusable model into a viable one.

2) Partial GPU Offload. The GTX 1650 Max-Q in our test system, despite lacking Tensor Cores, provides 4 GB GDDR5 VRAM. Partial layer offload using `-ngl 20` for Llama 3.2 3B would move approximately 20 of 32 transformer layers to GPU, reducing RAM pressure and leveraging GPU memory bandwidth (192 GB/s for GDDR5 vs 46 GB/s for DDR4). Based on the GPU offload test conducted separately for TinyLlama (which achieved 79.4 tok/s with full GPU offload vs 23.2 tok/s CPU-only), GPU acceleration provides approximately a 3–5× throughput improvement even on entry-level discrete GPUs without Tensor Cores.

3) Thread Count Tuning. The default behavior of using all available threads is not always optimal. On hyperthreaded CPUs like the i5-10300H (4 physical cores, 8 logical threads), using 4 threads (`-t 4`) instead of 8 can improve throughput by 10–20% for compute-bound workloads, as hyperthreading adds competition for shared execution resources during the memory-intensive decoding phase. Practitioners should benchmark thread counts of 4, 6, and 8 for their specific CPU.

4) Mmap and Memory Locking. llama.cpp's default memory-mapped file loading (`--mmap`) allows the OS to manage weight paging, which can introduce latency on HDDs. Adding `--no-mmap` forces weights into RAM on load, increasing load time but improving consistent inference speed by eliminating page fault overhead during generation. On SSD-equipped systems, mmap may be preferable; on HDD systems (as in our 1 TB HDD test platform), `--no-mmap` combined with `--mlock` (to prevent OS from swapping model weights) can improve sustained throughput by 15–30%.

5) Batch Size Optimization for Throughput. For batch processing applications where multiple prompts are processed sequentially, increasing the prompt batch size (`-b 512` or higher) improves prompt processing throughput significantly — in our tests, prompt processing reaches 30 tok/s for TinyLlama Q2_K. Pipelining prompt processing of the next query while generating the current response can

improve overall pipeline throughput by 20–40% for batch workloads.

6) Model Selection by Task. Not all tasks require the largest viable model. For factual retrieval and simple generation, TinyLlama Q2_K at 23.2 tok/s is the optimal choice. For tasks requiring precise instruction following or multi-step reasoning, Llama 3.2 1B with reduced context (`-ctx-size 2048`) provides better quality at 12.8 tok/s with significantly lower RAM usage than its default configuration. Maintaining a model registry that selects the appropriate model per task type can double effective system throughput compared to using a single large model for all tasks.

7) Quantization-Aware Model Selection. Our ablation study demonstrates that Q2_K is preferable to Q4_K_M for throughput on DDR4-constrained hardware. More broadly, practitioners should benchmark multiple quantization levels for their target model rather than defaulting to Q4_K_M. For very small models (sub-2B parameters), Q2_K may provide a better throughput-quality trade-off. For larger models (3B+), Q4_K_M's improved precision may be necessary to maintain acceptable output quality, making the GPU offload strategy (point 2 above) more important.

D. Limitations and Threats to Validity

This study has several limitations that should be considered when interpreting results. First, only two prompt categories were tested; a comprehensive evaluation would include code generation, multilingual prompts, long-document summarization, and few-shot learning tasks, each of which may produce different performance profiles. Second, output quality was not quantitatively evaluated in this study — perplexity scores, MMLU accuracy, and task-specific benchmark results (e.g., on HumanEval for code or GSM8K for reasoning) would provide a complete picture of the quality-efficiency trade-off across quantization levels.

Third, all experiments were conducted on a single hardware platform. The relative performance of models may differ on AMD Ryzen processors (which use a different memory controller and cache hierarchy), Apple Silicon (which uses unified memory with different bandwidth characteristics), or ARM-based edge devices. Fourth, the thermal behavior of the i5-10300H under sustained inference load was not characterized — long-running tasks may exhibit throughput degradation due to thermal throttling, particularly in thin-and-light laptop form factors. Fifth, the HDD storage in our test platform creates unusually slow model load times compared to SSD-equipped systems; reported load times should not be generalized to SSD deployments.

VI. CONCLUSION AND FUTURE WORK

This paper presented the first systematic benchmark study of five quantized lightweight LLMs deployed exclusively on CPU-based consumer laptop hardware with 8 GB RAM, providing empirical evidence to answer a question of significant practical importance: which models can actually run useably on the hardware that most people own.

Our core finding is clear and actionable: sub-1B parameter models quantized to Q2_K or Q4_K_M format are fully viable for real-time local inference on 8 GB RAM hardware, achieving 19–23 tok/s within a 1 GB memory footprint. This performance level supports interactive applications including chatbots, writing assistants, coding helpers, and offline document analysis tools — all without cloud dependency, internet connection, or GPU. The TinyLlama 1.1B Q2_K model, in particular, emerges as the optimal choice for constrained edge deployment, offering the best throughput, lowest RAM usage, and fastest cold-start time of any model tested.

The quantization ablation analysis reveals an important and non-obvious insight: on memory-bandwidth-constrained consumer hardware, more aggressive quantization (Q2_K) can outperform higher-precision alternatives (Q4_K_M) in throughput. This occurs because reduced weight size decreases memory bandwidth demand per decoding step, a bottleneck that dominates performance on DDR4-constrained systems. This finding challenges the common assumption that higher quantization quality always translates to better system performance.

A critical finding for practitioners considering Llama 3.2 deployment is the RAM amplification caused by large context windows. The 128K token context window in Llama 3.2 1B and 3B causes KV cache pre-allocation that consumes 4–5 GB of RAM regardless of actual inference context length, making these models impractical for 8 GB RAM deployment in their default configuration. The --ctx-size 2048 optimization flag resolves this issue, potentially transforming Llama 3.2 1B into one of the most capable models deployable within a 1.5 GB RAM budget. This optimization, combined with partial GPU offload for available discrete GPUs, represents the most impactful configuration changes for edge deployment.

Looking at the broader picture, our results validate the edge AI deployment hypothesis: meaningful, general-purpose AI inference is achievable on consumer hardware without cloud dependency. The combination of efficient quantized model formats, optimized CPU inference engines, and careful model selection creates a viable pathway to private, offline, and accessible AI applications running directly on end-user devices.

Future work will extend this study in several directions. GPU offload characterization using the GTX 1650 Max-Q will complete the CPU+GPU hybrid deployment picture. Output quality evaluation via perplexity measurements and task-specific benchmarks (MMLU, HumanEval, GSM8K) will quantify the quality-efficiency trade-offs across quantization levels. Extension to a broader hardware range

including AMD Ryzen laptops and Apple Silicon will test generalizability of our findings. Finally, evaluation of newer model families including Mistral 3B, Gemma 2 2B, and Qwen 1.5B will keep the benchmark current as the lightweight LLM landscape continues to evolve rapidly.

. ACKNOWLEDGMENT

The author thanks the open-source contributors to llama.cpp, Hugging Face, and the model developers at Meta AI, Microsoft Research, and the TinyLlama project for making their models and inference tools publicly available. This research was conducted independently.

LIST OF FIGURES

Fig. 1. Generation Throughput (tokens/sec) across models.

Fig. 2. End-to-End Latency across models.

Fig. 3. Peak RAM Consumption across models.

LIST OF TABLES

TABLE I. Experimental Hardware and Software Configuration

TABLE II. Master Benchmark Results — CPU-Only Inference (avg of 6 runs per model)

TABLE III. Per-Prompt Detailed Results — Avg, Min, Max Latency and Throughput (3 runs each)

TABLE IV. Quantization Ablation — TinyLlama 1.1B Q2_K vs Q4_K_M

TABLE V. Deployment Classification and Recommended Use Cases

. REFERENCES

- [1] T. Brown et al., "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems* (NeurIPS), vol. 33, pp. 1877–1901, 2020.
- [2] K. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," *Communications of the ACM*, vol. 63, no. 12, pp. 54–63, Dec. 2020.
- [3] G. Gerganov, "llama.cpp: LLM Inference in C/C++," GitHub Repository, 2023. [Online]. Available: <https://github.com/ggml-org/llama.cpp>
- [4] B. Jacob et al., "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *Proc. IEEE/CVF CVPR*, pp. 2704–2713, 2018.
- [5] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers," *arXiv preprint arXiv:2210.17323*, Oct. 2022.

- [6] J. Lin et al., "AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration," arXiv preprint arXiv:2306.00978, Jun. 2023.
- [7] T. Chen et al., "MLC-LLM: Universal LLM Deployment for Everyone," GitHub Repository, 2023. [Online]. Available: <https://github.com/mlc-ai/mlc-llm>
- [8] Y. Song et al., "PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU," arXiv preprint arXiv:2312.12456, Dec. 2023.
- [9] Y. Alizadeh et al., "LLM in a Flash: Efficient Large Language Model Inference with Limited Memory," arXiv preprint arXiv:2312.11514, Dec. 2023.
- [10] S. Laskar, "Benchmarking Large Language Models on Raspberry Pi and Low-Power Edge Devices," in Proc. ACM EdgeSys Workshop, 2023.
- [11] H. Wu, J. Guo, and W. Hu, "Efficient On-Device LLM Inference: A Survey of Mobile Deployment Strategies," arXiv preprint arXiv:2401.09896, Jan. 2024.
- [12] P. Zhang, G. Zeng, T. Wang, and W. Lu, "TinyLlama: An Open-Source Small Language Model," arXiv preprint arXiv:2401.02385, Jan. 2024.
- [13] Microsoft Research, "Phi-2: The Surprising Power of Small Language Models," Microsoft Research Blog, Dec. 2023. [Online]. Available: <https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/>
- [14] Meta AI, "Llama 3.2: Lightweight and Multimodal Models," Meta AI Blog, Sep. 2024. [Online]. Available: <https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/>
- [15] S. Sheng et al., "FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU," in Proc. ICML, 2023.
- [16] T. Dettmers et al., "SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression," arXiv preprint arXiv:2306.03078, Jun. 2023.