EnergyLLM: Dynamic LLM Edge Inference Team 7

Haebin Do
Harvard University
haebin_do@g.harvard.edu

Kevin He
Harvard University
kevinhe@g.harvard.edu

Angelica Kim
Harvard University
angelicakim@g.harvard.edu

Alexander Ingare
Harvard University
zingare@g.harvard.edu

***Abstract*—Deploying Large Language Models (LLMs) on edge devices is hindered by strict thermal and power constraints. This report presents EnergyLLM, a dynamic runtime scheduler designed to optimize inference performance under these conditions. Our approach utilizes a Group Relative Policy Optimization (GRPO) Reinforcement Learning (RL) agent to adaptively control GPU prefill and decode frequencies, batch sizes, and model depth based on real-time thermal and power headroom under Service Level Objective (SLO) targets. Unlike static baselines and built-in regulators, EnergyLLM balances competing objectives of throughput, end-to-end latency, energy, and accuracy. Experimental results on a Nvidia Jetson AGX Orin demonstrate that our RL scheduler achieves a 87.5% improvement in end-to-end latency and 47% power savings compared to max frequency, max batch size configuration with standard Dynamic Voltage and Frequency Scaling (DVFS) in thermally constrained environments.**

I. Introduction

As on-device LLMs promise to transform applications such as live translation and personal assistants, delivering efficient, low-latency inference at the edge demands hardware-aware scheduling and optimization strategy. Edge deployments must operate under tight power budgets (2–50 W) and stringent thermal limits (up to 100°C) without active cooling—conditions under which traditional GPU frequency governors perform poorly. These governors drive accelerators at maximum speeds until thermal throttling occurs, and they lack awareness of key properties of LLM inference, such as the distinct compute characteristics of the prefill and decode phases or the value of adjusting batch sizes of requests and model depth based on application context.

We propose EnergyLLM, a hardware-and-model-agnostic dynamic runtime scheduler that uses GRPO to adaptively configure GPU frequencies, batch sizes, and early exit layers based on real-time thermal and power headroom (see Fig. 1). Our key contributions include: (1) a cheap GRPO-based RL scheduler; (2) an adaptive reward function balancing throughput and energy efficiency based on available headroom; (3) dynamic early exit for accuracy-energy trade-offs; and (4) evaluation on Nvidia Jetson AGX Orin demonstrating effective thermal management.

II. Approach

We formulate the dynamic LLM inference scheduling problem as a Markov Decision Process (MDP). The state space $\mathcal{S}$ is 4-dimensional: $s_t = [f_p, f_d, b, h_T] \in [0,1]^4$, where f_p and f_d are normalized prefill and decode GPU frequency bins, b is normalized batch size, and h_T is temperature headroom computed as $h_T = \max(0, (T_{\max} - T)/T_{\max})$. The action space $\mathcal{A}$ consists of all combinations of discrete frequency bins (11 values each for prefill and decode), batch sizes (32) resulting in $|\mathcal{A}| = 11 * 11 * 32 = 3,872$ possible actions. We create a separate model for early exit with an action space of $|\mathcal{A}| = 11 * 11 * 6 * 8 = 5,808$ (early exit layers 9-16, bins 1-31 incrementing by 6). Each action $a_t = (f_p^{(a)}, f_d^{(a)}, b^{(a)}, d^{(a)})$ directly sets the GPU frequency for prefill and decode phases, the batch size for the next inference batch and number of layers for early exit.

The objective is to find a policy $\pi_\theta(a|s)$ that maximizes expected per-step reward, where constraints are enforced through penalty terms in the reward function:

$$\max_\theta \mathbb{E}_{s\sim\rho, a\sim\pi_\theta}\left[R(s,a)\right] \tag{1}$$

where ρ is the state distribution induced by the policy, and $R(s,a)$ is the reward function that balances throughput, energy efficiency, and constraint violations. The reward function incorporates exponential penalties for temperature, latency, power and entropy violations, effectively converting the constrained optimization into an unconstrained problem where constraint satisfaction is incentivizes through the reward structure.

A. GRPO Algorithm

We employ Group Relative Policy Optimization (GRPO) [1] to train our policy, a reinforcement learning algorithm that eliminates the need for a value function by computing advantages through group-based comparisons. Our policy network π_θ consists of 4 fully-connected layers with hidden dimension 32, ReLU activations, and outputs action logits over the discrete action space.

B. Reward Function

Our reward function balances performance objectives and constraint satisfaction based on thermal and power headroom:

$$R(s,a) = \underbrace{\begin{cases} \tilde{E} & \text{if } H_{\text{eff}} \le 0.33 \\ H_{\text{eff}} \cdot \tilde{T} + (1 - H_{\text{eff}}) \cdot \widetilde{\text{EDP}} \cdot \tilde{T} & \text{otherwise} \end{cases}}_{\text{performance metric}} - \underbrace{(P_{\text{SLO}} + P_{\text{temp}} + P_{\text{power}} + P_{\text{entropy}})}_{\text{constraint penalties}} \tag{2}$$

where $\tilde{T}$, $\tilde{E}$, and $\widetilde{\text{EDP}}$ are normalized throughput, energy efficiency, and energy-delay product. The performance metric adapts to thermal conditions: when effective headroom—the weighted combination of thermal and power headrooms—falls below $H_{\text{eff}} \le 0.33$ (corresponding to device junction temperatures 80°C), the scheduler switches into an energy-

preserving mode that prioritizes efficiency. Above this threshold, the reward smoothly balances throughput and EDP in proportion to the available headroom.

The penalty terms enforce latency, temperature, power, and early-exit constraints. Each $P_{\text{constraint}}$ increases only when its corresponding metric exceeds its allowable limit, ensuring policy learns to avoid violations through higher performance scores. For example, the entropy penalty regulates early-exit decisions by penalizing high-entropy exits at low temperatures. These components enable the agent to transition smoothly between high-performance and energy-preserving regimes under strict thermal and power limits.

C. Dynamic Early Exit

While scaling batch size results in lower energy per request, it also reduces average token throughput since fewer prompts will be processed in parallel. To offset this, we propose using early exits during LLM inference (i.e. reducing model depth) to dynamically scale a model at runtime and reduce the memory transfer and compute requirements per token. Unlike other model compression techniques like quantization and pruning which rely on specialized hardware units to achieve energy and latency gains, early exit simply stops the forward pass of an input before all transformer layers are complete. The output embeddings from this intermediate layer are projected on the LM head to choose the next token. Early exit delivers clear reductions in energy and latency (Figure 9, 10), without overheads for dequantization or non-optimal memory access patterns for pruning. Early exit also scales easily at runtime without the need to re-format model weights. Layers not used during early exit are not loaded from memory to the GPU.

We demonstrate this approach with the Llama3.2 1B LayerSkip [2], which was trained with layer dropout and an early exit loss function to specifically preserve accuracy at lower intermediate layers. We verify these improved accuracy claims compared to the standard Llama3.2 1B model on the Wikitext, HellaSwag, COPA, and ARC-Easy datasets, where we see a significant decrease in perplexity for Wikitext and increase in accuracy for HellaSwag, COPA, and ARC-Easy at the same early exit layers on both models (Figure 12). We integrate early layer exit into our RL environment by letting the agent choose between layers 9-16 during its action selection.

III. Implementation

A. Jetson Environment

We collected power measurements from a *Nvidia Jetson AGX Orin* with 32 GB of memory using the Llama3.2 1B and Llama3.2 1B LayerSkip models. We separated and sampled 1000 prompts from the Google Natural Questions dataset. Power consumption is measured using the built in `tegrastats` tool, which provides real-time readings of GPU/SOC power, CPU power, memory power, 5V rail power, and GPU temperature measurements. These measurements are sampled at 1 ms intervals during inference. We turn on and off the built-in fan to model devices that use passive cooling.

B. RL Training

The training process consists of two phases: offline pre-training and online fine-tuning. Offline pre-training uses supervised learning to match the policy's action distribution to a target distribution derived from reward-weighted action frequencies in grid search data, providing a strong prior before online interactions. Online fine-tuning collects transitions through real hardware interactions, periodically updating the policy (every 200 steps) using group-relative advantages. The reference policy is synchronized every 1000 steps to maintain training stability. Figure 1 presents the online training procedure.

IV. Results

We evaluate the overhead introduced by the RL agent across three dimensions: search cost, action selection and policy updates. We also measure the performance of the static optimal configuration identified via grid search and compare GRPO's performance against DVFS.

A. Search Overhead

We compare the search overhead of GRPO against an exhaustive grid search for identifying optimal configurations. Figure 7 shows that GRPO reduces energy consumption, power draw, and latency by approximately 80% relative to grid search. Moreover, GRPO achieves an approximately 100% reduction in cumulative regret—the difference between the optimal achievable reward and the reward of the learned policy over time. Figure 2 also shows that our GRPO agent explores a much smaller action space yet converges to the same optimal configuration. These results highlight the efficiency of GRPO's online learning approach and demonstrate its practicality in deployment settings where search cost is critical.

B. RL Agent Policy Update Overhead

We observe negligible runtime overhead from executing the RL agent at each inference stp. Action selection contributes less than 0.1% of total latency and energy, while online policy updates account for less than 3%. This difference reflects the computational requirements of a full forward and backward pass during policy updates, including optimizer steps, compared to the lightweight forward pass used for action selection.

To further reduce the overhead of policy updates, we replaced the single output layer with a factorized action head. The original output layer contains $11 \times 11 \times 32 \times 32 = 123{,}904$ (single head) parameters. In contrast, the factorized design uses three independent heads whose combined dimensionality yields $(11 + 11 + 32) \times 32 = 3{,}872$, resulting in a dramatic reduction of memory footprint of the output-layer by 96.8% and lowers the RL policy updates overhead by more than 48% (Figure 4). The resulting accuracy degradation was negligible (less than 3%) which is within normal evaluation variance. With a larger hidden dimension, the accuracy loss becomes completely negligible.

C. Performance Comparison of Static Configurations

We compare static configurations to demonstrate that simply maximizing perforance parameters leads to suboptimal results. The optimal static configuration identified through grid search under low thermal regimes outperforms the maximum static configuration (batch size 32, maximum GPU frequency) across multiple metrics: 10.8% energy savings, 12.1% EDP savings, 1.5% latency reduction, 8.6% throughput improvement, and 2.5% temperature reduction (See Figure 3). This shows that fixed maximum parameters cause thermal inefficiencies and suboptimal resource utilization.

However, static configurations cannot adapt to changing thermal conditions. GRPO's dynamic scheduling provides additional benefits by continuously adjusting batch size and frequencies based on real-time thermal headroom, enabling high performance during favorable conditions while preventing thermal violations and crashes when constraints tighten, as shown in comparison with DVFS in section IV-D.

D. Performance Comparison to DVFS

We compare our GRPO scheduler against a DVFS baseline that uses a static maximum configuration (batch size 32 and maximum GPU frequency). As shown in Figure 5, the baseline system quickly reaches the $100°$C thermal limit and ultimately crashes, whereas GRPO maintains safe operating temperatures by proactively reducing batch size and modulating GPU frequencies when thermal headroom tightens.

Under high-temperature conditions, GRPO achieves a 87.5% improvement in end-to-end latency at the first timestep corresponding to the max temperatures (DVFS: $99°$C, GRPO: $92°$C, respectively). GRPO keeps the latency at approximately 4s while the baseline hits 7.5s at $99°$C and eventually exceeds the 15s SLO as Jetson fails to regulate the temperature. This improvement arises from GRPO's ability to preserve higher GPU frequencies by reducing batch size during thermal stress, in contrast to DVFS, which collapses to minimum frequencies once throttling is triggered. GRPO also reduces power consumption by 47% through phase-aware adjustments to GPU frequency during both prefill and decode.

E. Early Exit

Dynamic early exit adds a fourth control dimension to our scheduler, allowing the RL agent to trade off model accuracy for energy efficiency and token throughput. With this added control dimension, the GRPO agent results in 9.8% and 19.2% faster prefill throughput than static and DVFS, 4.7% and 8.9% faster decode throughput than static and DVFS, and 29.3% and 33.1% faster total request time than static and DVFS when the device is at $99°$C. Early exit uses an average of 15.05 layers with 18.7% using 12 layers (Figure 13).

TABLE I
EARLY EXIT INFERENCE COMPARISON AGAINST THE BASELINES AT $99°$C

Metric	Static	DVFS	GRPO
Prefill Throughput (tokens/sec)	3380.76	3114.62	**3711.58**
Decode Throughput (tokens/sec)	507.27	487.82	**531.18**
Total Request Time (sec)	6.7262	7.1075	**4.7563**

V. Related Works

GreenLLM is a LLM serving system that optimizes prefill and decode frequency on A100 nodes to achieve better datacenter energy efficiency by staying closer to the SLO targets. They use a simple dual-loop feedback controller to balance decode token throughput and frequency [3]. In the edge domain, *Camel* [4] addresses the trade-off between latency and energy on resource-constrained devices (Jetson AGX Orin) by modeling the selection of GPU frequency and batch size as a Multi-Armed Bandit problem to minimize the Energy-Delay Product (EDP). *CLONE* [5] proposes a hardware-software co-design that combines offline generative model pruning with an online reinforcement learning-based DVFS controller to optimize per-token energy efficiency under real-time constraints.

These approaches rely on static model compression and require additional hardware support or lack dynamic adaptation to thermal saturation. To date, our research is the first GRPO-based LLM scheduler that adaptively balances token throughput and energy throughout the temperature range for edge devices, while also dynamically adapting the model size with early exit.

VI. Conclusion

We present EnergyLLM, a lightweight GRPO-based dynamic runtime scheduler for edge LLM inference that adaptively manages GPU frequencies, batch sizes, and early exit layers under thermal constraints. Our key contribution is demonstrating that a cheap, hardware-and-model-agnostic RL scheduler can achieve significant performance improvements with minimal overhead.

Key lessons learned: (1) reward design directly dictates agent behavior—our adaptive headroom-based formulation guides the agent to prioritize energy efficiency under thermal pressure while maximizing throughput when headroom is available; (2) KL tuning governs training stability; and (3) exposing additional dynamic control knobs significantly improves optimization.

Our system achieves the project goal of delivering scalable and energy efficient LLM inference on edge devices without exceeding thermal thresholds. The scheduler's hardware and LLM independence makes it broadly applicable across different edge platforms and model architectures. Future work includes multi-model support, enabling the scheduler to manage concurrent inference requests across different model architectures while still maintaining shared thermal constraints. Additionally, EnergyLLM could be extended to a broader range of edge-compute devices to verify the heterogeneous nature of our hardware-aware RL approach. We could also explore combining dynamic early exit layers with quantization and pruning to further maximize energy efficiency in extremely resource-constrained environments.

VII. Group Contribution Statement

Haebin Do and Angelica Kim co-developed the core energy and latency efficient RL pipeline from pre-training to deployment, designing the valid hyper-parameters, state/action

space and multi-objective reward functions to create multiple performance comparison results against Static Grid search and DVFS. Kevin He conducted baseline comparisons, measured the accuracy/latency/energy tradeoffs of early exit, implemented the early-exit action, and helped design the entropy penalty. Alexander Ingare designed the dynamic early-exit inference strategy, implemented the entropy penalty, and ran performance experiments with the early exit scheduling.

REFERENCES

[1] Z. Shao et al., *Deepseekmath: Pushing the limits of mathematical reasoning in open language models*, 2024. arXiv: 2402.03300 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2402.03300.

[2] M. Elhoushi et al., "Layerskip: Enabling early exit inference and self-speculative decoding," *arXiv preprint*, 2024, FAIR at Meta, GenAI at Meta, Reality Labs at Meta. [Online]. Available: https://github.com/facebookresearch/LayerSkip.

[3] Q. Liu, D. Huang, M. Zapater, and D. Atienza, *Greenllm: Slo-aware dynamic frequency scaling for energy-efficient llm serving*, 2025. arXiv: 2508.16449 [cs.PF]. [Online]. Available: https://arxiv.org/abs/2508.16449.

[4] H. Xu et al., *Camel: Energy-aware llm inference on resource-constrained devices*, 2025. arXiv: 2508.09173 [cs.NI]. [Online]. Available: https://arxiv.org/abs/2508.09173.

[5] C. Tian et al., "Clone: Customizing llms for efficient latency-aware inference at the edge," in *Proceedings of the 2025 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '25, Boston, MA, USA: USENIX Association, 2025, ISBN: 978-1-939133-48-9.

APPENDIX

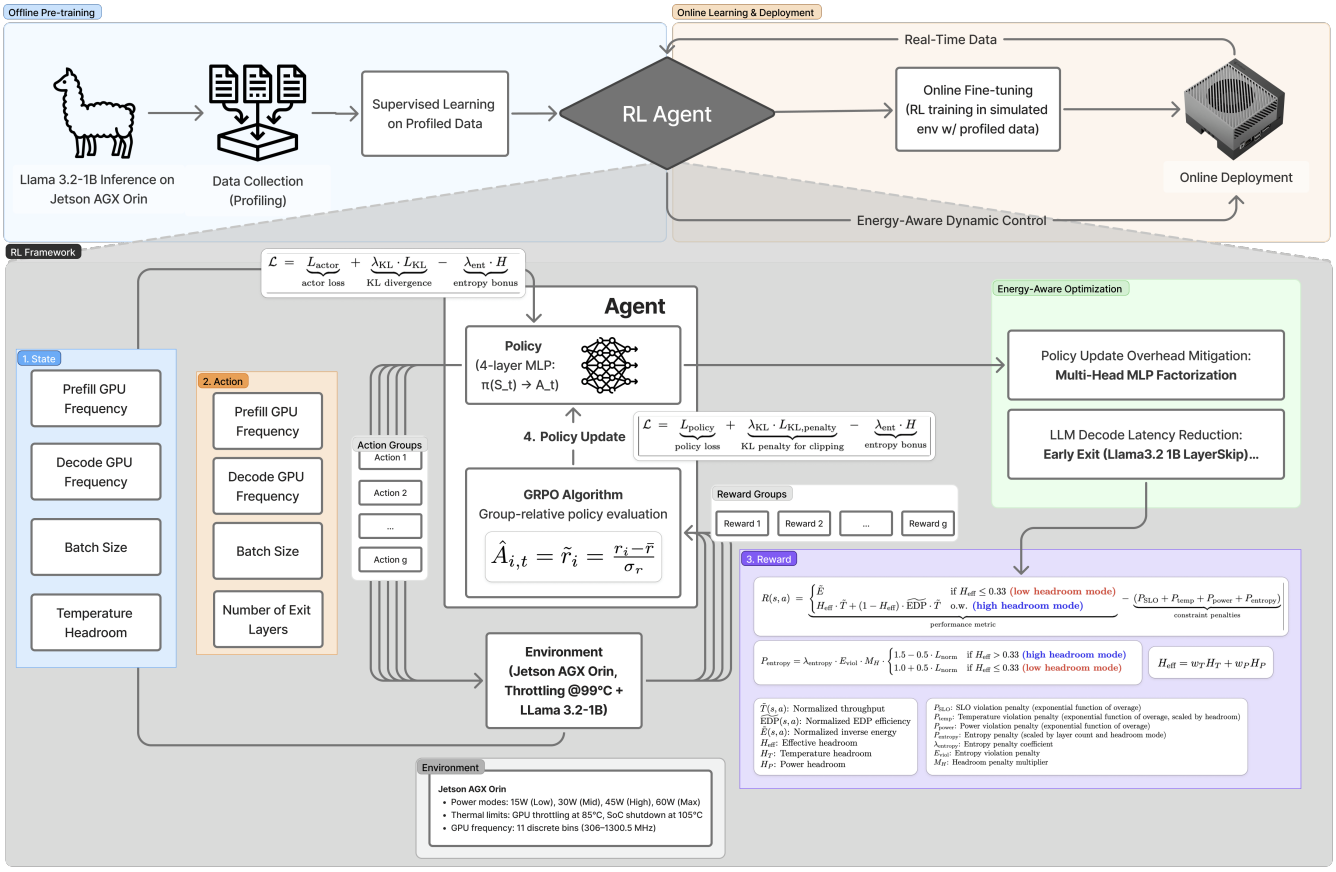

Fig. 1. Overview of the EnergyLLM RL agent pipeline.

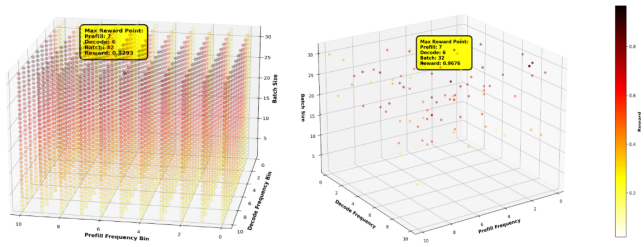

Fig. 2. Search-space exploration patterns for GRPO and grid search.

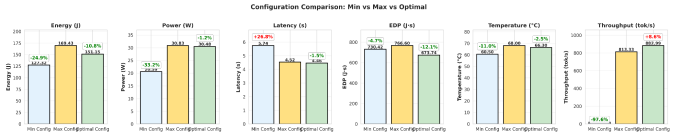

Fig. 3. Comparison of static optimal configuration against minimum and maximum configurations.

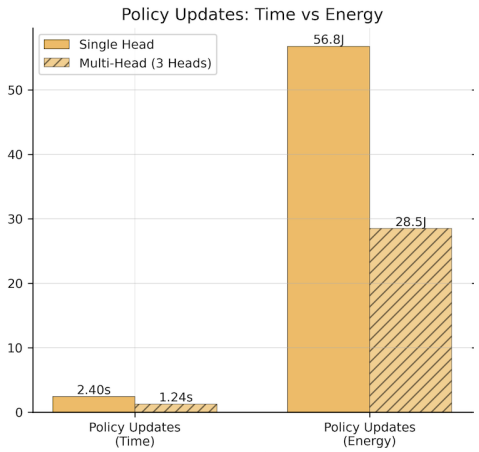

Fig. 4. Effect of factorizing the MLP output head on policy update cost.

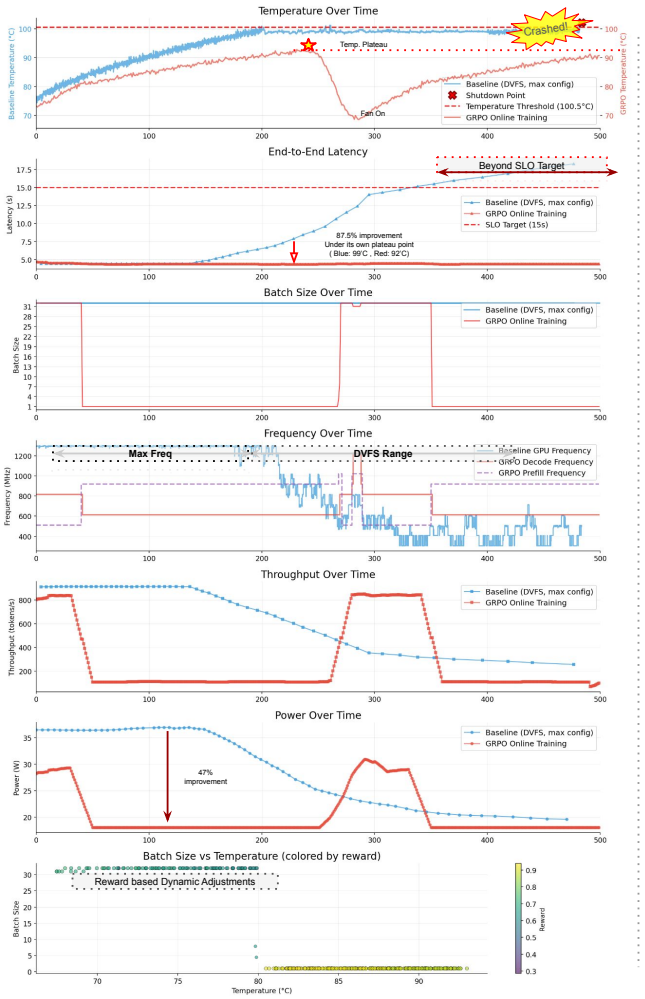

Fig. 5. GRPO performance improvements over DVFS across throughput, latency, power, and temperature.

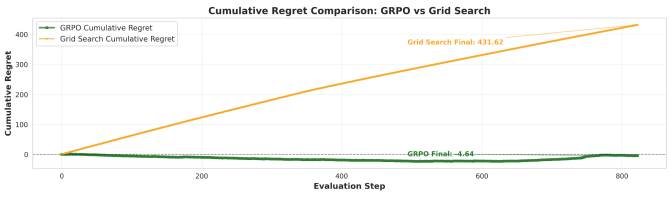

Fig. 6. Cumulative regret of GRPO compared with grid search.

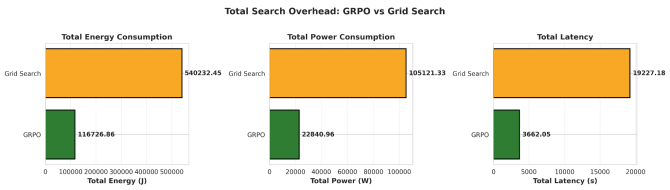

Fig. 7. Search overhead of GRPO compared with grid search.

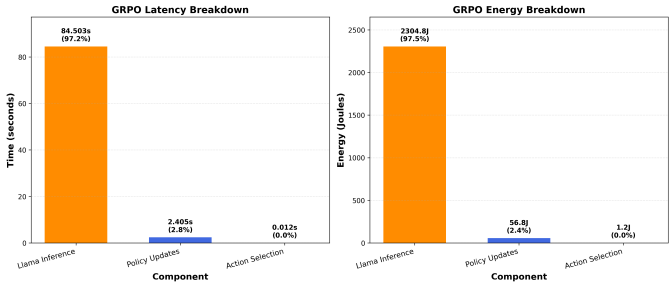

Fig. 8. RL-agent overhead: latency (left) and energy (right). LLM inference dominates total energy consumption.

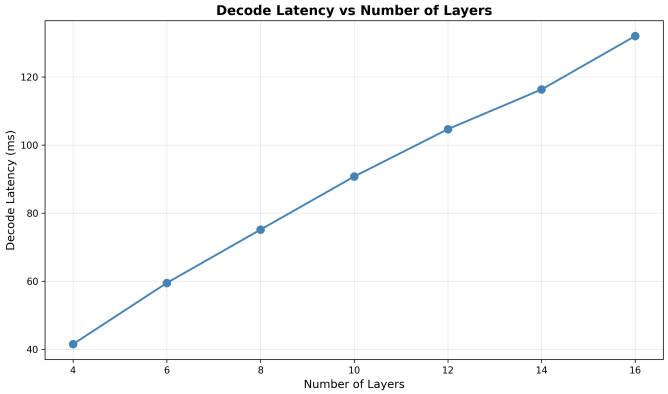

Fig. 9. Effect of early exit layers on decode latency. Since early exit simply chooses an early transformer layer to exit at, we see a direct linear relationship between layers used (model depth) and decode latency per token.

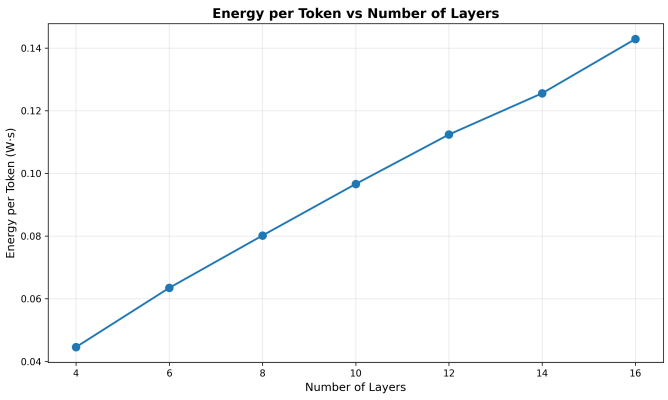

Fig. 10. Effect of early exit layers on energy per token. LayerSkip exhibits a direct linear relationship with energy per token.

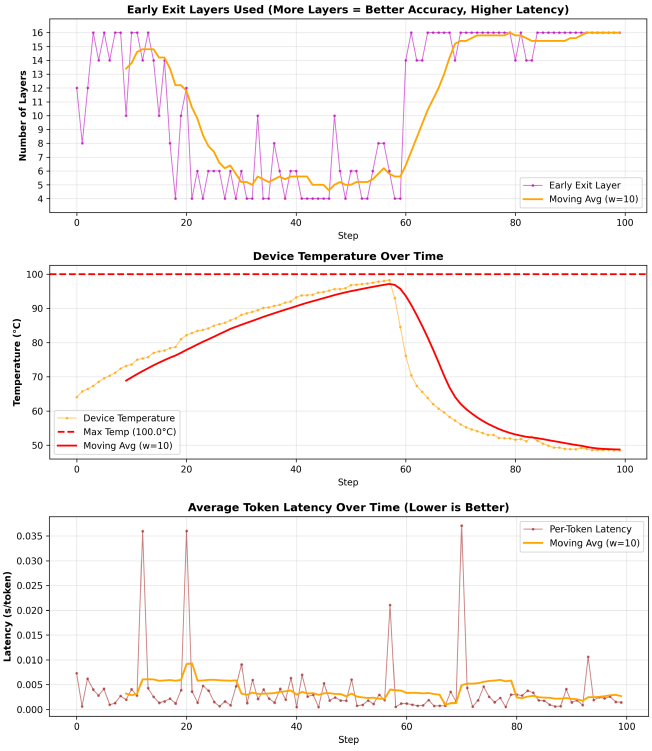

Fig. 11. Average token latency results for dynamic early exit layers ranging from 4-16 layers. As the device temperature headroom shrinks and approaches it's threshold, the reward function entropy penalty begins to encourage the RL agent to swap from full layer usage to low layer usage. Conversely, as temperature shrinks from the limits and reaches suitable operation ranges, the reward function encourages the agent to switch to using full layers for maximum accuracy again. Throughout this switching, the average token latency stays constant over time, indicating the token-latency saving effects of LayerSkip.

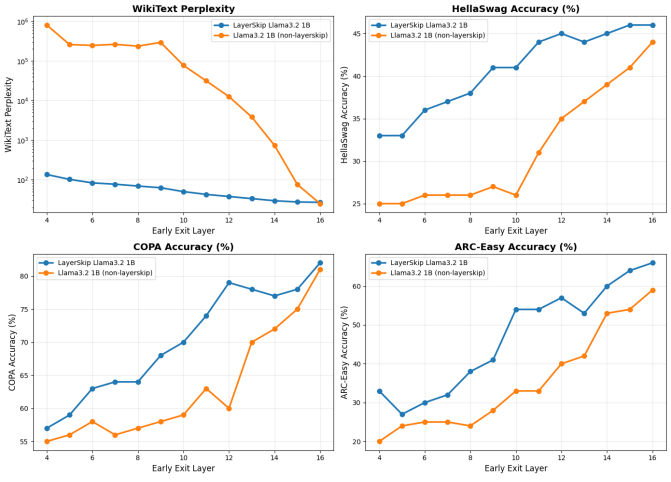

Fig. 12. Comparison of LayerSkip Llama3.2 1B vs non-LayerSkip for early exit layer perplexity on WikiText dataset and accuracy on HellaSwag, COPA, and ARC-Easy datasets. In our RL agent configuration with LayerSkip Llama3.2 1B, we see a 5% accuracy drop on HellaSwag and 14% accuracy drop on COPA when going from 16 to 9 layers.

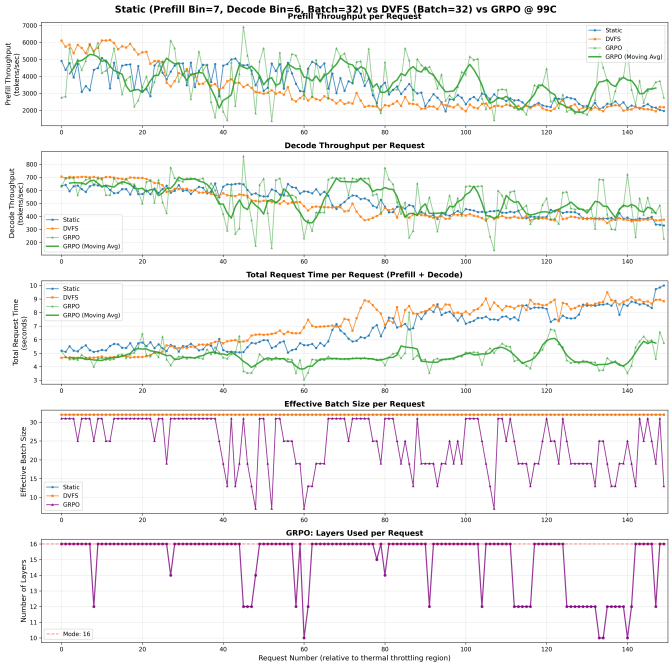

Fig. 13. Comparison of GRPO vs an optimal static frequency/batch size (prefill_bin=7 & decode_bin=6) configuration vs DVFS with a batch size of 32. The GRPO agent dynamically adjusts the number of layers used as batch size decreases to keep prefill and decode throughput high.