APPENDIX

A Additional Platform Information

In §2.3, we briefly introduced CPU, GPU, and memory governors on mobile devices. Here we provide more detailed information on these governors.

GPU governor. The Quickstep (AOSP, 2022b) governor is used to manage the GPU frequency in recent generations of phones such as the Google Pixel family from Pixel 6 to Pixel 9. It determines the target frequency based on a predefined table of minimum and maximum utilization for each GPU frequency which is provided by the manufacturers. Fig. 12 depicts the threshold limits on Pixel 7 and Pixel 7 Pro devices (found from `dvfs_table`)—if the utilization falls below the minimum utilization, the frequency is scaled down and if it exceeds the maximum utilization, the frequency is scaled up.

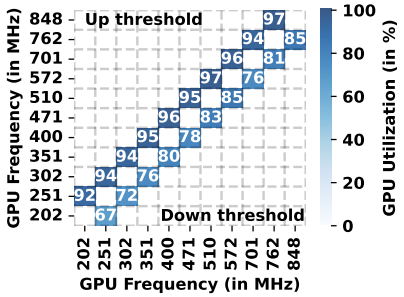


Figure 12: Target utilization range per GPU frequency

Memory governor. The memory interface (MIF) (ARM, 2012; 2015; Conway, 2013) connects the main memory to all the system components such as the CPU and GPU. The number of read and write transactions with the main memory is affected by MIF frequency, *e.g.,* if the MIF frequency is too low then the CPU will incur higher data latency and will start stalling. As running tasks switch between compute and memory phases, there will be bursts of memory transactions interleaved with idle or lightly loaded periods. The main objective of MIF is to deliver data with minimum latency during such burst periods. To this end, MIF uses an interactive governor which increases memory frequency to the peak when it observes high memory bus utilization, and steps down the frequency as the utilization drops, by calculating a target frequency based on a formula and a predefined factor (AOSP, 2022a) provided by the manufacturer, and setting the MIF frequency to one of the 13 frequencies from Table 1 closest to the target frequency. The above process is repeated every 20 ms (Chan, 2012; Saber, 2015; Banerjee & John, 2018; Banerjee, 2018).

CPU governor. Android smartphones employ Energy-aware scheduling (EAS) (kernel.org, 2015) for CPU power management, which encompasses both task placement and DVFS control. Deciding the right frequency for a given task is done in two steps. (1) First, EAS determines the *load* of the task. A straightforward estimation of a task's load is its CPU utilization. However, such an estimation is not frequency-invariant, as the CPU utilization is usually higher under lower frequency. To this end, EAS applies different scaling factors to the CPU utilizations measured under different frequencies, such that the scaled loads for the same task stay roughly the same. The current load of a task is estimated from its historical loads sampled every millisecond, with earlier samples exponentially decayed. (2) Second, EAS is provided a per-cluster load-to-frequency lookup table that is used to find the lowest frequency that can satisfy the estimated task load. The task load continues to decay when the task is waiting for I/O or other resources, *e.g.,* GPU (Corbet, 2013). Thus, EAS may choose lower CPU frequencies for GPU-heavy tasks like LLM inference due to the low task load.

Table 1: Available frequencies in Pixel 7 and Pixel 7 Pro

Governor	Available Frequencies (MHz)
CPU	500, 851, 984, 1106, 1277, 1426, 1582, 1745, 1826, 2048, 2188, 2252, 2401, 2507, 2630, 2704, 2802, 2850
GPU	151, 202, 251, 302, 351, 400, 471, 510, 572, 701, 762, 848
Memory	421, 546, 676, 845, 1014, 1352, 1539, 1716, 2028, 2288, 2535, 2730, 3172

B Additional Model Information

In §6, we evaluated the performance of CORE with a set of popular LLM models. Here we show the detailed model information in Table 2.

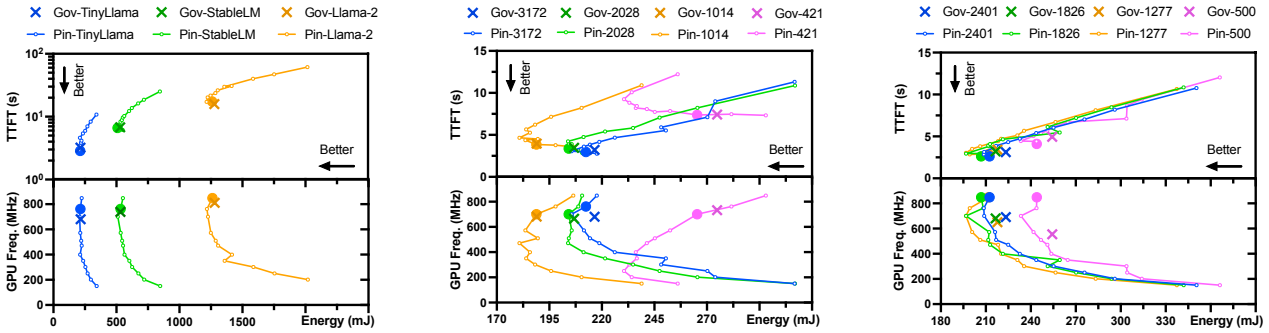
Table 2: Evaluated decoder models in llama.cpp.

Model	#Layers	Hidden size	Size	Device
TinyLlama (Zhang et al., 2024a)	22	2048	1.1B	Pixel 7
DeepSeek-R1-Distill-Qwen (DeepSeek-AI, 2025)	28	1536	1.5B	Pixel 7
Smallcloudai Refact-fim	32	2048	1.6B	Pixel 7
StableLM-Zephyr	32	2560	2.7B	Pixel 7
Microsoft Phi-2	32	2560	2.7B	Pixel 7 Pro
Meta Llama-2 (Touvron et al., 2023)	32	4096	6.7B	Pixel 7 Pro

C Additional GPU Governor Results

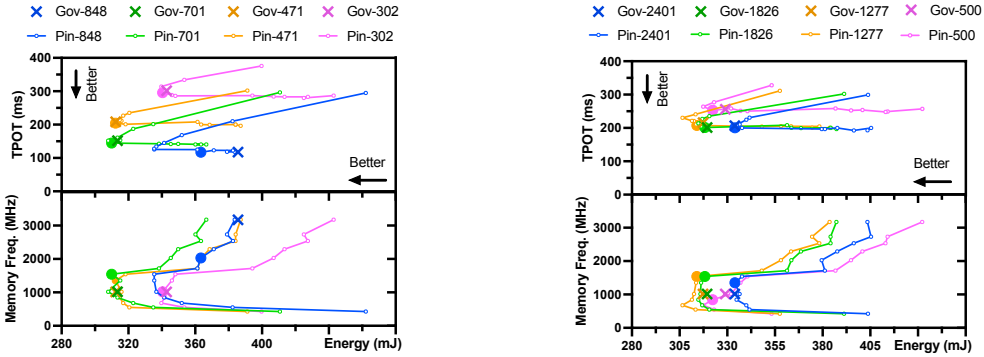
In §5.1, we analyzed GPU governor's impact on LLM inference by pinning CPU and memory frequencies for the decode stage. Here we show the results for the prefill stage.

Prefill. The results are shown in Fig. 13. We make the following observations. (1) Fig. 13(a) upper shows unlike decode, the GPU governor achieves close to optimal TTFT and energy-per-token for the three LLM models. This is because compared to decode, prefill enjoys higher GPU utilization due to token batching, which leads the GPU governor to choose sufficiently high frequencies, as shown in Fig. 13(a) lower half. For instance, the effective GPU frequencies are 680.7, 738.8, and 811.3 MHz for TinyLlama,



(a) Various Models (b) Varying pinned f_{MEM} (c) Varying pinned f_{CPU}

Figure 13: Prefill TTFT and energy-per-token of the GPU governor (`Gov`, marked with ×) compared with pinning the GPU at each available frequency (`Pin`). We set f_{CPU}=1826 MHz in (a) and (b), and f_{MEM}=3172 MHz in (a) and (c). Plots in (b) and (c) are for TinyLlama. The lowest-latency frequency combinations with the same energy drain as the GPU governor, `Pin-Opt`, is marked with ●.



(a) Varying pinned f_{GPU} (b) Varying pinned f_{CPU}

Figure 14: Decode latency and energy drain of the memory governor (`Gov`, marked with ×) compared with pinning memory at each available frequency (`Pin`). GPU and CPU frequencies are pinned at 471 MHz and 1826 MHz. Plots in (b) and (c) are for TinyLlama. The lowest-latency frequency combinations with the same energy drain as the memory governor, `Pin-Opt`, is marked with ●.

StableLM, and Llama-2, respectively; pinning the GPU frequency to the optimal frequency 762 MHz for TinyLlama only reduces the GPU governor's TTFT by 11.2%, from 3.2 to 2.9 seconds. (2) Fig. 13(b) shows the GPU governor also achieves close to optimal TTFT and energy efficiency with various pinned memory frequencies, again from choosing high frequencies, *e.g.,* the effective GPU frequencies for 2028, 1014, and 421 MHz pinned memory frequencies are 664.8, 682.1, and 733.3 MHz, respectively. (3) Fig. 13(c) shows the GPU governor can achieve close to optimal TTFT and energy efficiency by choosing high GPU frequencies with various pinned CPU frequencies.

D Additional Memory Governor Results

In §5.2, we analyzed memory governor's impact on LLM inference for various models with one pair of pinned GPU and CPU frequencies. Here we focus on TinyLlama and show the results with varying pinned GPU frequency (Fig. 14(a)) or pinned CPU frequency (Fig. 14(b)) for the decode stage.

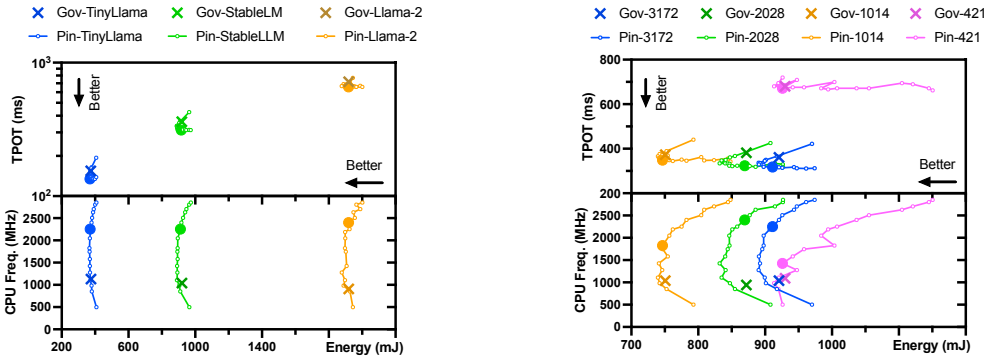
We see the memory governor achieves near optimal inference latency and energy consumption per token. One exception is observed in Fig. 14(b), when the GPU is pinned at its highest frequency of 848 MHz. The memory governor constantly runs at the highest memory frequency (3172 MHz) in this case, resulting in high energy consumption (385.6 mJ per token and 118.2 ms TPOT) compared to the lowest-energy pinned memory frequency of 2028 MHz (363.4 mJ per token) with the almost same TPOT (117.3 ms) as the memory governor.

E Additional EAS Governor Results

In §5.3, we analyzed EAS's impact on LLM inference for StableLM with various pinned GPU frequencies for the decode stage. Here we first show additional results for the decode stage.

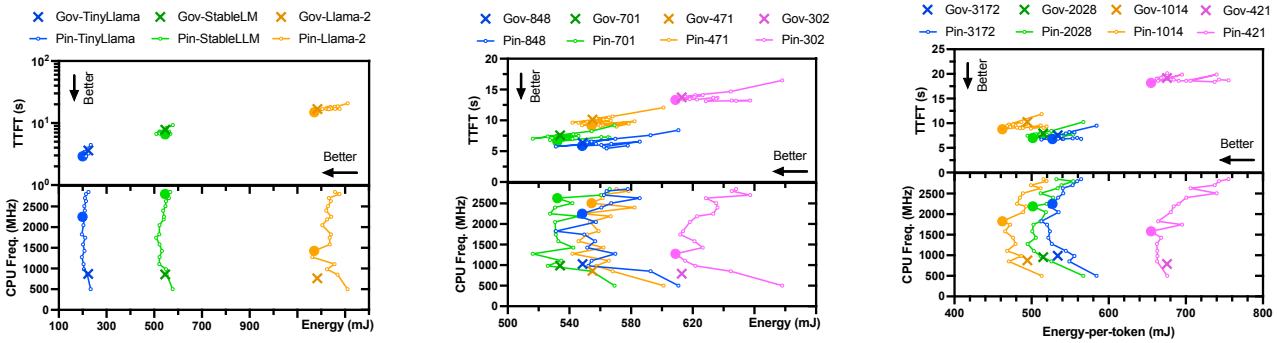
Decode. For the decode stage, similar to the observations made in §5.3, EAS consistently achieves higher TPOT and energy consumption compared to optimal pinned CPU frequency, regardless of model sizes (Fig. 15(a)) or pinned memory frequencies (Fig. 15(b)).

Next, we show the results for the prefill stage.



(a) Various Models (b) Varying pinned f_{MEM}

Figure 15: Decode latency and energy drain of EAS (`Gov`, marked with ×) compared with pinning CPU at each available frequency (`Pin`). We set $f_{MEM} = 3172$ MHz in (a), and $f_{GPU} = 701$ MHz in (a) and (b). Plot (b) is for StableLM. The lowest-latency frequency combinations with the same energy drain as EAS, `Pin-Opt`, is marked with ●.



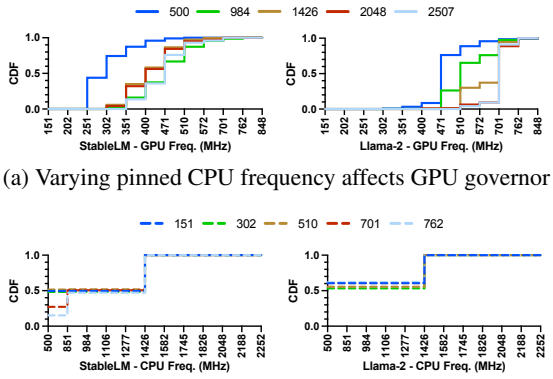
(a) Various Models (b) Varying pinned f_{GPU} (c) Varying pinned f_{MEM}

Figure 16: Prefill latency and energy drain of EAS (`Gov`, marked with ×) compared with pinning the CPU at each available frequency (`Pin`). We set $f_{MEM} = 3172$ MHz in (a) and (b), and set $f_{GPU} = 701$ MHz in (a) and (c). Results in (b) and (c) are for StableLM. `Pin-Opt` is marked with "●".

Prefill. For the prefill stage, similar to the decode stage, EAS consistently achieves higher TTFT (for the same energy budget) or energy consumption (for the same TTFT), regardless of model sizes (Fig. 16(a)), pinned GPU frequencies Fig. 16(b), or pinned memory frequencies (Fig. 16(c)), due to consistently choosing low frequencies in all settings, as shown in the lower half of Fig. 16. For instance, the TTFT under EAS for TinyLlama, StableLM, and Llama-2 are 3.6, 7.8, and 16.8 seconds, which can be reduced by 18.9%, 16.2%, and 11.2% by pinning CPU to the optimal frequencies of 2802, 2802, and 1426 MHz with a similar energy consumption, respectively. The longer TTFT under EAS can be explained by the fact that EAS chooses overly low CPU frequencies. Specifically, the effective CPU frequencies for three models are 870.7, 858.2, and 763.4 MHz.

F Additional Antagonistic Effect Results

In §5.4 (Fig. 8), we demonstrated the antagonistic effect between the EAS and GPU governors in real time by pinning and unpinning the GPU frequency.



(a) Varying pinned CPU frequency affects GPU governor

(b) Varying pinned GPU frequency affects EAS

Figure 17: CDF of the amount of time that GPU (upper) or CPU (lower) runs at a certain frequency under GPU governor or EAS, respectively. Results are collected in the decode phase with StableLM (left) and Llama-2 (right) served by llama.cpp. We set f_{MEM}=3172 MHz.

To further illustrate the antagonistic effect between the EAS and GPU governors, we let the default GPU governor con-

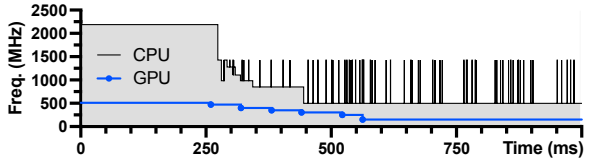


Figure 18: Runtime trace of CPU and GPU frequencies showing the antagonistic effect.

trol GPU frequencies throughout the experiment and pin the CPU to 2188 MHz at the beginning, then unpin it at 250 ms, *i.e.,* let the default EAS governor control CPU frequencies. As illustrated in Fig. 18, immediately after the CPU is unpinned, the CPU governor drops its frequency from 2188 to 984 MHz at 287 ms. During this period, the GPU frequency is at 471 MHz. At 320 ms, the GPU frequency drops from 471 to 400 MHz, which in turn drives the CPU governor to lower the CPU frequency from 984 to 851 MHz at 343 ms. The antagonistic effect continues, ultimately driving the CPU governor to lower its frequency to its minimum of 500 MHz at 445 ms, and the GPU governor to lower its frequency to its minimum of 151 MHz at 563 ms.

In additional to Fig. 7, Fig. 17(a) shows that lowering the pinned CPU frequency leads to GPU governor lowering the GPU frequencies chosen regardless of model sizes. Fig. 17(b) shows that lowering the pinned GPU frequency leads to EAS lowering the CPU frequencies chosen regardless of model sizes.

G Additional Evaluation Results for Goal G2

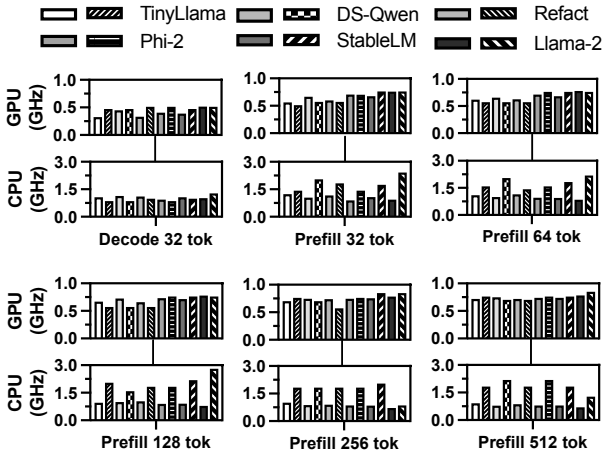


Figure 19: CPU/GPU frequency comparison of CORE with `Gov` (the default governors), for goal **G2** (minimize energy under latency target). Latency target is the TTFT or TPOT under the default governors.

Fig. 19 shows the CPU/GPU frequencies that correspond to the energy-per-token results previously presented in Fig. 10b. We see that while inferencing with the same TTFT as `Gov`, CORE reduces the energy-per-token by searching and setting the CPU/GPU to the optimal frequency combination. For example, while prefilling 32 tokens with DeepSeek-R1-Distill-Qwen (shortened as DS-Qwen), by setting the GPU and CPU frequencies to 572 and 2048 MHz (compared to 663.4 and 1045.6 MHz with `Gov`), CORE reduces the energy-per-token by 10.3% (from 310.3 to 278.2 mJ) while inferencing at the same or lower TTFT (5.6 s with `Gov` and 5.4 s with CORE). While decoding 32 tokens with DS-Qwen, by setting the GPU and CPU frequencies to 471 and 851 MHz (compared to 448.5 and 1134.5 MHz with `Gov`), CORE reduces the energy-per-token by 7.8% (from 459.0 to 423.3 mJ) while inferencing at the same or lower TPOT (346.8 ms with `Gov` and 298.4 ms with CORE).