# EOP-LLM: Energy Oriented Pruning for Large Language Models

**Guan Huang and Tao Shu**[†]
Department of Computer Science and Software Engineering
Auburn University
Auburn, AL 36849, USA
{gzh0040, tshu}@auburn.edu

## Abstract

In deployed large language models (LLMs), inference energy consumption has grown rapidly and has emerged as a key bottleneck in large-scale deployment, yet most existing inference efficiency methods focus on reducing FLOPs or latency, rather than explicitly modeling or enforcing end-to-end inference energy constraints. We propose EOP-LLM, an energy-oriented dynamic pruning framework that enables LLM inference under explicit per-sequence energy budgets. EOP-LLM combines a device-calibrated energy proxy with lightweight token and feed-forward (FFN) selectors, coordinated through a global dual variable, to dynamically allocate computation while preserving model quality. Extensive experiments on LLaMA 3.2 (1B/3B) and LLaMA 3.1 (8B) demonstrate that EOP-LLM consistently outperforms state-of-the-art dynamic pruning baselines under matched energy budgets, while strictly adhering to per-sequence energy constraints.

## 1 Introduction

Over the past few years, large language models (LLMs) have rapidly transitioned from research prototypes to widely deployed systems supporting applications such as search engines (Xiong et al., 2024), virtual assistants (Liang and Tong, 2025), and enterprise analytics tools (Kumar et al., 2025). As these deployments expand, their electricity consumption has emerged as a critical challenge. In 2024, U.S. data centers supporting AI workloads consumed an estimated 183 TWh of electricity, accounting for more than 4% of national demand; moreover, this demand is forecast to rise to approximately 426 TWh by 2030, a level that would exceed the United Kingdom's most recent annual electricity consumption (Leppert, 2025; DESNZ, 2025). Meanwhile, LLM-based services remain far from saturation. For example, a Pew Research Center survey reports that only 34% of U.S. adults have ever used ChatGPT (Sidoti and McClain, 2025), indicating that substantial growth in energy consumption from LLM deployment is still ahead.

Recent studies demonstrate that inference in deployed LLM services has already surpassed training as the dominant and fastest-growing source of energy consumption (Argerich and Patiño-Martínez, 2024; Jegham et al., 2025). Unlike training, which is performed once and amortized over a model's lifetime, inference begins the moment a model is deployed and runs continuously across large data center fleets, serving massive volumes of user queries and downstream applications; leading services now process billions of requests per day (Niu et al., 2025). As a result, the cumulative energy cost of inference far exceeds the one-time cost of training and continues to rise with the growing scale of LLM usage. Consistent with this trend, prior work reports that inference accounts for up to 70% of Meta's AI power consumption, approximately 60% of Google's ML energy usage, and 80–90% of AWS's cloud computing demand (Fernandez et al., 2025; Jin et al., 2025). Therefore, developing LLM inference mechanisms that can reduce and control energy consumption while maintaining reliable, high-quality outputs has become a critical challenge for practical LLM deployment.

Although inference energy has emerged as a dominant bottleneck in LLM deployment, most existing work on LLM inference efficiency still primarily focuses on optimizing FLOPs, rather than explicitly modeling or regulating inference energy consumption (Wan et al., 2023; Xiao et al., 2023; Zhu et al., 2024; Zhou et al., 2024; Liu et al., 2024; Fang et al., 2025). Broadly, these methods fall into two categories: static model compression and dynamic computation. Static model compression reduces inference cost (FLOPs) by replacing the original large model with a smaller, fixed surrogate model. These methods typically prune weights, channels, or attention heads using activation or gra-

[†]Corresponding author.



*Findings of the Association for Computational Linguistics: ACL 2026*, pages 7443–7464

dient based importance scores, apply post-training quantization to reduce numerical precision, or distill knowledge from a large teacher into a compact student model (Frantar et al., 2022; Ma et al., 2023; Zhang et al., 2024; Muralidharan et al., 2024; Huang et al., 2024; Guo et al., 2025). In contrast, dynamic computation methods aim to reduce FLOPs by tailoring the executed computation to each input at inference time. Typically, these methods prune uninformative tokens or key–value (KV) cache entries, skip layers or attention blocks for inputs estimated to require less computation, or gate attention and feed-forward network (FFN) components based on signals derived from intermediate hidden states (Anagnostidis et al., 2023; Fu et al., 2024; Le et al., 2025; Long et al., 2025; Tao et al., 2025; Zhao et al., 2025; Luo et al., 2025).

At first glance, FLOP-driven static and dynamic methods may appear adequate for reducing inference energy, since they reduce inference-time computation, and FLOPs are often treated as proxies for energy. It is therefore natural to expect that tuning pruning ratios would yield energy-efficient behavior on a given device. In practice, this intuition does not hold. FLOP reductions correlate only weakly with actual energy consumption, which also depends on memory traffic, kernel execution behavior, and other system-level overheads that vary substantially across layers and token positions (Tian et al., 2024; Wang et al., 2025). As a result, FLOP-driven methods cannot reliably translate pruning decisions into predictable energy savings.

Additionally, practical deployment conditions further limit the applicability of FLOP-based methods for reliably reducing inference energy. In AI data centers, the energy headroom available to an LLM can shift rapidly due to device, rack, or cluster-level power caps, co-located workloads, and thermal throttling, while incoming queries vary widely in their computational demands (James, 2025; Kearney, 2025). Meanwhile, incoming queries vary widely in length and computational difficulty, resulting in substantial per-request variation in inference energy. Without explicit request-level energy control, computation is often wasted on less informative inputs while insufficiently allocated to more challenging ones, leading to inefficient energy utilization and avoidable degradation in model quality. Even worse, on mobile and other edge platforms, instantaneous power and thermal limits are substantially more restrictive, severely constraining per-query computation and leading to increased latency or even infeasible inference. Clearly, an inference mechanism that can both reduce inference energy and explicitly respect per-sequence energy constraints while preserving model quality is needed.

Orthogonal to FLOP-driven approaches, several recent works incorporate measured energy into their method design to reduce runtime energy, either by guiding serving-level scheduling decisions (e.g., device placement and cache management) or by deriving fixed compressed models offline (Wang et al., 2025; Tian et al., 2024). However, these approaches do not provide a model-internal mechanism for enforcing explicit per-sequence energy budgets during inference, which is our focus.

In this work, we propose EOP-LLM (Energy-Oriented Pruning for LLMs), a dynamic inference framework that treats energy as a primary constraint and enforces explicit per-sequence energy budgets on a given device while maintaining high-quality outputs. While energy headroom varies widely across hardware platforms and fluctuates over time in data centers due to power caps and co-located workloads, the core of EOP-LLM is a device-calibrated energy model that predicts the joule cost of a forward pass as a function of attention and FFN operations and associated memory accesses, with hardware-specific coefficients obtained from short on-device power measurement sweeps. This formulation enables per-sequence energy-constrained inference, in which each input is executed subject to an explicit device-level energy budget. Building on this energy model, EOP-LLM augments a pretrained decoder-only LLM with two lightweight selectors that jointly determine which tokens are processed and how much FFN capacity is allocated at each layer, explicitly orienting inference-time computation around per-sequence energy budgets while preserving output quality. Specifically, the token selector removes low-importance token blocks and compacts the KV cache to reduce attention and memory costs, whereas the FFN selector chooses layer widths that best align with learned importance scores under the remaining FFN energy allotment. Across LLaMA 3.2 (1B/3B) and LLaMA 3.1 (8B) under varying per-sequence energy budgets, EOP-LLM consistently outperforms state-of-the-art dynamic pruning baselines, achieving up to a 7.4% average accuracy gain on commonsense reasoning at 50% of dense execution energy, while adhering to the specified per-sequence budgets.

## 2 Problem Statement

We consider a pretrained decoder-only transformer deployed on a target device $q$. Let $f_\theta$ denote the model with parameters $\theta$ and $L$ transformer layers. We evaluate on a dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, where $x^{(i)} = (x_1^{(i)}, \ldots, x_{P_i}^{(i)})$ is an input prompt and $y^{(i)} = (y_1^{(i)}, \ldots, y_{T_i}^{(i)})$ is a reference (ground-truth) continuation. For each sampled pair $(x, y)$, we write $P := |x|$ and $T := |y|$, where $|\cdot|$ denotes sequence length. For evaluation under teacher forcing, the conditional likelihood of a continuation given a prompt can be defined as:

$$p_\theta(y \mid x) = \prod_{t=1}^{T} p_\theta(y_t \mid x, y_{<t}), \; y_{<t} := (y_1, \ldots, y_{t-1}). \quad (1)$$

**Inference-time execution control.** Inference consists of a prefill stage on the prompt $x$ followed by autoregressive decoding. We use $t = 0$ to denote prefill (with $y_{<0} := \emptyset$) and $t \in \{1, \ldots, T\}$ to denote decoding steps that generate $y_t$. The corresponding prefix length is defined as:

$$n_t := \begin{cases} P, & t = 0, \\ P + t - 1, & t \in \{1, \ldots, T\}. \end{cases} \quad (2)$$

We model inference-time computation control via the following layerwise execution decisions.

*(i) Token retention and KV compaction.* For each layer $\ell$ and stage $t$, let $\mathcal{S}_{\ell,t} \subseteq \{1, \ldots, n_t\}$ denote the token positions kept and executed at layer $\ell$, with $n_{\ell,t}^{\text{keep}} := |\mathcal{S}_{\ell,t}|$. Self-attention is computed on the packed subsequence indexed by $\mathcal{S}_{\ell,t}$, and only these tokens are stored in the KV cache for subsequent decoding.

*(ii) FFN width selection.* Each layer $\ell$ has a position-wise FFN with intermediate width $d_{\text{ff}}^{(\ell)}$ and model hidden dimension $d$. At step $t \in \{0, 1, \ldots, T\}$, we select a set of active FFN channels $\mathcal{C}_{\ell,t} \subseteq \{1, \ldots, d_{\text{ff}}^{(\ell)}\}$, with $w_{\ell,t}^{\text{keep}} := |\mathcal{C}_{\ell,t}| \in \mathcal{W}_\ell$. This selection determines the effective FFN width executed at layer $\ell$ and step $t$. For standard two-projection FFNs with weights $W_1^\ell \in \mathbb{R}^{d_{\text{ff}}^{(\ell)} \times d}$ and $W_2^\ell \in \mathbb{R}^{d \times d_{\text{ff}}^{(\ell)}}$, this corresponds to slicing $\widetilde{W}_{1,t}^\ell = W_1^\ell[\mathcal{C}_{\ell,t}, :]$ and $\widetilde{W}_{2,t}^\ell = W_2^\ell[:, \mathcal{C}_{\ell,t}]$. For gated FFNs (e.g., GeGLU (Raffel et al., 2020) and SwiGLU (Touvron et al., 2023)), the same $\mathcal{C}_{\ell,t}$ is applied to all relevant input projections.

**Execution policies.** These execution decisions are selected by a policy $\pi \in \mathcal{Q}$ that, at each stage $t \in \{0, 1, \ldots, T\}$, maps the available prefix information to layerwise execution choices.

**Device-calibrated energy model.** We focus on steady-state inference after warm-up, where model weights and runtime state reside in GPU memory or host DRAM, as in modern LLM serving systems (Kwon et al., 2023; Verma and Vaidya, 2023). In this setting, we obtain energy measurements on device $q$ using standard power telemetry interfaces, including NVML (NVIDIA, 2024) for GPU power and RAPL (Colmant and Marcuzzi, 2023). for CPU and DRAM energy, and fit a device-calibrated predictor $\widehat{E}_q$ that estimates the energy (in Joules) consumed when executing $f_\theta$ under a policy $\pi$. Specifically, given the execution decisions $\{\mathcal{S}_{\ell,t}, \mathcal{C}_{\ell,t}\}$ produced by $\pi$, with induced sizes $n_{\ell,t}^{\text{keep}} = |\mathcal{S}_{\ell,t}|$ and $w_{\ell,t}^{\text{keep}} = |\mathcal{C}_{\ell,t}|$, we model the total inference energy as an additive decomposition across layers and inference steps:

$$\widehat{E}_q(x, y; \pi) = \sum_{t=0}^{T} \sum_{\ell=1}^{L} \widehat{E}_{q,\ell,t}\Big(n_t, n_{\ell,t}^{\text{keep}}, w_{\ell,t}^{\text{keep}}\Big) + \gamma_q, \quad (3)$$

where $\gamma_q$ captures all execution overheads that are independent of the execution decisions.

**Energy-constrained inference objective.** Given a pretrained model $f_\theta$ deployed on device $q$, an evaluation dataset $\mathcal{D}$, and a per-sequence energy budget $E_{\text{bud}}(x, T)$, the goal of EOP-LLM is to learn an execution policy $\pi \in \mathcal{Q}$ that preserves output quality while respecting the energy constraint. Energy-constrained inference is then formulated as:

$$\min_{\pi \in \mathcal{Q}} \mathbb{E}_{(x,y)\sim\mathcal{D}}\Big[-\sum_{t=1}^{T} \log p_{\theta,\pi}(y_t \mid x, y_{<t})\Big] \quad \text{s.t. } \widehat{E}_q(x, y; \pi) \le E_{\text{bud}}(x, T), \quad \text{for all } (x, y) \in \mathcal{D}. \quad (4)$$

## 3 Methods

### 3.1 Architecture Overview

EOP-LLM is a dynamic inference framework designed to solve the energy-constrained inference problem in Eq. 4 by explicitly regulating per-sequence energy consumption during inference, without retraining or fine-tuning the pretrained backbone $f_\theta$. Specifically, EOP-LLM processes each input sequence through a sequence of coordinated execution stages. First, given the current prefix $(x, y_{<t})$, the framework estimates the energy cost of candidate execution choices using a device-calibrated energy predictor $\widehat{E}_q$. Next, based on this estimate, EOP-LLM identifies and retains informative prefix tokens, compacting the KV cache to reduce attention computation and memory costs. Then, under the remaining energy budget, EOP-LLM allocates FFN channels across layers by selecting effective layer widths that best support the retained tokens. After that, the model executes the transformer layers using the compacted attention context and the selected FFN widths, producing the

next-token distribution. This process is repeated autoregressively for subsequent decoding steps. The pseudocode is provided in Appendix B.

### 3.2 Device-Calibrated Energy Model

We now instantiate the device-calibrated energy model $\widehat{E}_q(x, y; \pi)$ defined in Eq. 3. The key idea is to model measured joules using memory-traffic and arithmetic cost terms, enabling explicit accounting of how token retention and FFN width selection affect energy consumption.

**Blockwise Retention.** Rather than making fine-grained token or channel level decisions, we adopt a blockwise formulation that groups tokens and FFN channels into contiguous execution units. This choice is dictated by GPU execution granularity: modern GPU kernels operate on fixed-size tiles and warps, and pruning below this granularity does not reliably alter kernel execution, as kernels continue to load full tiles and perform the same memory accesses (Dao et al., 2022; NVIDIA, 2025). Consequently, fine-grained pruning may mask values but fails to reduce actual computation or energy consumption in practice.

Token positions at stage $t$ are partitioned into contiguous blocks of size $B$. Let $\mathcal{B}_t := \{1, \ldots, \lceil n_t/B \rceil\}$ index the blocks, and let $\mathcal{I}_{b,t} := \{(b-1)B+1, \ldots, \min(bB, n_t)\}$ denote the token indices in block $b$. To make the later execution policy trainable via gradient-based optimization, we introduce relaxed token-block execution gates $g_{\ell,b,t} \in [0, 1]$ for each layer $\ell$ and block $b$. These relaxed gates induce a continuous retained length at layer $\ell$ and stage $t$ as:

$$\bar{n}_{\ell,t} = \max\Big\{\min(B, n_t),\ \sum_{b \in \mathcal{B}_t} |\mathcal{I}_{b,t}|\, g_{\ell,b,t}\Big\}, \tag{5}$$

which ensures that at least one token block is executed. Similarly, the intermediate FFN dimension $d_{\text{ff}}^{(\ell)}$ of layer $\ell$ is partitioned into contiguous channel blocks of size $G$, indexed by $\mathcal{J}_\ell$. Each block $j \in \mathcal{J}_\ell$ corresponds to channel indices $\mathcal{K}_{\ell,j} \subseteq \{1, \ldots, d_{\text{ff}}^{(\ell)}\}$ with $|\mathcal{K}_{\ell,j}| = G$. To make the execution policy trainable, we introduce relaxed channel-block execution gates $m_{\ell,j,t} \in [0, 1]$, which induce the effective FFN width:

$$\bar{w}_{\ell,t} = \max\Big\{\min(G, d_{\text{ff}}^{(\ell)}),\ \sum_{j \in \mathcal{J}_\ell} |\mathcal{K}_{\ell,j}|\, m_{\ell,j,t}\Big\}. \tag{6}$$

Note that $\bar{n}_{\ell,t}$ and $\bar{w}_{\ell,t}$ are continuous surrogates induced by relaxed gates to enable gradient-based optimization; at deployment time, execution uses their hardened counterparts, as described later.

**Energy proxy.** Given an input $(x, y)$ and an execution policy $\pi$, the induced execution decisions determine the effective token counts and FFN widths $\{\bar{n}_{\ell,t}, \bar{w}_{\ell,t}\}_{\ell,t}$. Accordingly, the total inference energy (in joules) on device $q$ can be modeled as:

$$\begin{aligned}\widehat{E}_q(x, y; \pi) = \sum_{t=0}^{T} \sum_{\ell=1}^{L} \Big(&\alpha_{q,\ell}^{\text{HBM}}\, \text{Bytes}_{\ell,t}^{\text{att}} + \alpha_{q,\ell}^{\text{MAC}}\, \text{MAC}_{\ell,t}^{\text{att}} \\ &+ \beta_{q,\ell}^{\text{HBM}}\, \text{Bytes}_{\ell,t}^{\text{ffn}} + \beta_{q,\ell}^{\text{MAC}}\, \text{MAC}_{\ell,t}^{\text{ffn}}\Big) + \gamma_q.\end{aligned} \tag{7}$$

where $\text{Bytes}_{\ell,t}^{(\cdot)}$ and $\text{MAC}_{\ell,t}^{(\cdot)}$ are functions of $\bar{n}_{\ell,t}$ and $\bar{w}_{\ell,t}$, and denote the memory traffic and arithmetic operation counts, respectively, of the attention and FFN modules at layer $\ell$ and stage $t$. The coefficients $\{\alpha_{q,\ell}^{\text{HBM}}, \alpha_{q,\ell}^{\text{MAC}}, \beta_{q,\ell}^{\text{HBM}}, \beta_{q,\ell}^{\text{MAC}}, \gamma_q\}$ are trained by regression via short on-device calibration sweeps based on the measured energy, and the details are provided in Appendix D.2.

Note that, during KV-cached decoding, only the newly generated token is executed through the FFN. Accordingly, we define the executed-token count (per layer) as:

$$\bar{\kappa}_{\ell,t} := \begin{cases} \bar{n}_{\ell,0}, & t = 0 \quad \text{(prefill)}, \\ 1, & t \in \{1, \ldots, T\} \quad \text{(decode)}. \end{cases} \tag{8}$$

The count terms are then expressed in closed form as functions of the relaxed retained length $\bar{n}_{\ell,t}$ and the relaxed effective FFN width $\bar{w}_{\ell,t}$:

$$\begin{aligned} &\text{Bytes}_{\ell,t}^{\text{att}} = a_{0,\ell} + a_{1,\ell}\, \bar{n}_{\ell,t}, && \text{MAC}_{\ell,t}^{\text{att}} = u_{1,\ell}\, \mathbb{1}\{t \geq 1\}\, \bar{n}_{\ell,t} + u_{2,\ell}\, \mathbb{1}\{t = 0\}\, \bar{n}_{\ell,t}^2, \\ &\text{Bytes}_{\ell,t}^{\text{ffn}} = \bar{\kappa}_{\ell,t}\big(c_{0,\ell} + c_{1,\ell}\, \bar{w}_{\ell,t}\big), && \text{MAC}_{\ell,t}^{\text{ffn}} = v_{1,\ell}\, \bar{\kappa}_{\ell,t}\, \bar{w}_{\ell,t}, \end{aligned} \tag{9}$$

where $a_{\cdot,\ell}, u_{\cdot,\ell}, c_{\cdot,\ell}, v_{\cdot,\ell}$ are coefficients related to the model dimensions of $f_\theta$, whose exact definitions and the detailed procedure of determining their values are provided in Appendix C. The indicator $\mathbb{1}\{t = 0\}$ captures that quadratic attention cost arises only during prefill, whereas KV-cached autoregressive decoding incurs linear attention cost $O(\bar{n}_{\ell,t})$ (Zhao et al., 2024). For notational simplicity, we omit the device index $q$ on intermediate budget quantities, as we are focusing on device $q$.

### 3.3 Token and FFN Selectors

EOP-LLM parameterizes the execution policy $\pi$ using two lightweight selectors operating at complementary granularities. At stage $t$, the token selector produces layerwise token-block scores, and the FFN selector produces channel-block scores. These scores quantify relative importance and are used to derive energy-feasible token retention and FFN width decisions in subsequent steps.

**Token selector.** At stage $t$, given the available prefix of length $n_t$ and the token-block partition $\{\mathcal{I}_{b,t}\}_{b \in \mathcal{B}_t}$, we score token-block importance after a dense warm-up of $\ell_0$ layers. Let $H_{t,i}^{(\ell_0)} \in \mathbb{R}^d$ denote the hidden state at position $i$. To this end, we first summarize each token block by a compact feature vector:

$$\rho_{b,t} = \Big[\mathrm{mean}_{i\in\mathcal{I}_{b,t}} H_{t,i}^{(\ell_0)},\ \max_{i\in\mathcal{I}_{b,t}} H_{t,i}^{(\ell_0)},\ \mathrm{std}_{i\in\mathcal{I}_{b,t}} H_{t,i}^{(\ell_0)},\ h_{b,t},\ \mathrm{pos}_{b,t},\ \omega_{b,t}\Big]. \tag{10}$$

where $\mathrm{pos}_{b,t}$ encodes the block position, $\omega_{b,t}$ is a detached exponential moving average (EMA) of block-aligned activations, and $h_{b,t}$ captures local uncertainty. The uncertainty is defined as:

$$h_{b,t} = -\frac{1}{|\mathcal{I}_{b,t}|}\sum_{i\in\mathcal{I}_{b,t}}\sum_{k=1}^{K_{\mathrm{unc}}} p_{t,i}^{(k)}\log p_{t,i}^{(k)}, \qquad p_{t,i} = \mathrm{softmax}\Big(W_{\mathrm{unc}} H_{t,i}^{(\ell_0)}\Big), \tag{11}$$

where $W_{\mathrm{unc}}$ is a learned projection head and $K_{\mathrm{unc}}$ is a small design constant controlling the granularity of the uncertainty estimate.

Because token-block importance is context-dependent and influenced by neighboring prefix blocks, we model local block-to-block dependencies along the 1D chain of token blocks $\mathcal{B}_t$ using two rounds of message passing with depthwise separable dilated 1D convolutions. Let $\mathcal{P}_t^{(0)}$ stack $\{\rho_{b,t}\}_{b\in\mathcal{B}_t}$ row-wise. For $\tau = 0, 1$, we compute:

$$U_t^{(\tau)} = \mathrm{DWConv}\Big(\mathcal{P}_t^{(\tau)};\ \mathrm{dilation} = D_\tau\Big), \qquad \mathcal{P}_t^{(\tau+1)} = \mathrm{ReLU}\Big(U_t^{(\tau)} W_{1\times1}^{(\tau)} + \vartheta^{(\tau)}\Big), \tag{12}$$

where $U_t^{(\tau)}$ are aggregated block features, $D_\tau$ is the dilation rate, $W_{1\times1}^{(\tau)}$ is a channel-mixing projection, and $\vartheta^{(\tau)}$ is a learned bias vector.

Raw layerwise token-block scores are obtained by applying a per-layer linear head to the final features $\mathcal{P}_t^{(2)}$:

$$s_{\ell,t} = \mathcal{P}_t^{(2)}\nu_\ell + \iota_\ell\,\mathbf{1}, \qquad \ell\in[L], \tag{13}$$

where $s_{\ell,t}$ stacks $\{s_{\ell,b,t}\}_{b\in\mathcal{B}_t}$ and $\nu_\ell, \iota_\ell$ are learned per-layer parameters, and $\mathbf{1}$ is an all-ones vector. Such a definition of score is to map the context-enriched token-block features $\mathcal{P}_t^{(2)}$ into scalar importance scores using a minimal affine readout that preserves monotonic ranking while allowing layer-specific scaling and bias calibration. In subsequent steps, these scores are transformed into relaxed token-block gates $\{g_{\ell,b,t}\}_{b\in\mathcal{B}_t}$.

**FFN selector.** To score FFN channel-block importance, given the channel-block partition $\{\mathcal{K}_{\ell,j}\}_{j\in\mathcal{J}_\ell}$, at stage $t$, we first form a feature vector for block $j$ in layer $\ell$:

$$r_{\ell,j,t} = \Big[\underbrace{\|W_2^{(\ell)}[:,\mathcal{K}_{\ell,j}]\|_F\,\|W_1^{(\ell)}[\mathcal{K}_{\ell,j},:]\|_F}_{\text{static prior}},\ \underbrace{\mathrm{EMA}_t(\|\xi_{\ell,j,t}\|_2)}_{\text{activation usage}},\ \underbrace{\mathrm{EMA}_t(\|\nabla_{\xi_{\ell,j,t}}\mathcal{L}\|_2)}_{\text{Fisher proxy}}\Big]. \tag{14}$$

where $W_1^{(\ell)}$ and $W_2^{(\ell)}$ are the FFN projection matrices in layer $\ell$, $\|\cdot\|_F$ denotes the Frobenius norm, $\xi_{\ell,j,t}$ denotes the FFN intermediate activation restricted to indices $\mathcal{K}_{\ell,j}$, and $\mathcal{L}$ is the teacher-forcing loss used for selector training.

Raw channel-block scores are computed as:

$$z_{\ell,j,t} = \eta_\ell^\top r_{\ell,j,t}, \tag{15}$$

where $\eta_\ell$ is a per-layer projection vector. Such a definition is to convert the multi-signal FFN block descriptors $r_{\ell,j,t}$ into a single scalar importance score using a lightweight per-layer linear projection that preserves relative ranking.

### 3.4 Budget Coordination and Training

On device $q$, at stage $t$, the token selector outputs token-block scores $\{s_{\ell,b,t}\}_{\ell=1}^{L,\ b\in\mathcal{B}_t}$ and the FFN selector outputs channel-block scores $\{z_{\ell,j,t}\}_{\ell=1}^{L,\ j\in\mathcal{J}_\ell}$. To enforce a global per-sequence energy budget on the resulting execution decisions, we convert these scores into relaxed execution gates $\{g_{\ell,b,t}\}$ and $\{m_{\ell,j,t}\}$ by pricing them with a global dual variable $\lambda \geq 0$ and device-calibrated marginal energy costs from the energy proxy in Eq. 7.

**Stage budget accounting.** To enforce a per-sequence energy budget during autoregressive inference, we maintain a remaining proxy budget $E_t^{\mathrm{rem}}$ at each stage $t$. Because the proxy energy $\widehat{E}_q$ is additive across inference stages, this remaining budget can be updated incrementally. We account for the constant overhead once by initializing:

$$E_0^{\mathrm{rem}} = E_{\mathrm{bud}}(x,T) - \gamma_q,\ E_{t+1}^{\mathrm{rem}} = \Big[E_t^{\mathrm{rem}} - \widehat{E}_q^{(t)}(x,y;\pi)\Big]_+, \tag{16}$$

where $[u]_+ := \max\{0,u\}$ and $\widehat{E}_q^{(t)}(x,y;\pi) := \sum_{\ell=1}^{L}\widehat{E}_{q,\ell,t}(n_t,\bar{n}_{\ell,t},\bar{w}_{\ell,t})$. denotes the stage-$t$ energy under the proxy.

**Token gates.** We construct energy-oriented token gates by pricing token retention with the attention-side marginal energy of the proxy, holding FFN width fixed and priced separately. For layer $\ell$ at stage $t$, the per-token marginal attention cost is:

$$\Delta_{\ell,t}^{\mathrm{tok}} := \tfrac{\partial\widehat{E}_{q,\ell,t}^{\mathrm{att}}}{\partial\bar{n}_{\ell,t}} = \alpha_{q,\ell}^{\mathrm{HBM}}\,a_{1,\ell} + \alpha_{q,\ell}^{\mathrm{MAC}}\,(u_{1,\ell}\,\mathbf{1}\{t\geq1\} + 2u_{2,\ell}\,\mathbf{1}\{t=0\}\,\bar{n}_{\ell,t}), \tag{17}$$

where $\widehat{E}_{q,\ell,t}^{\mathrm{att}}$ is defined as:

$$\widehat{E}_{q,\ell,t}^{\mathrm{att}} := \alpha_{q,\ell}^{\mathrm{HBM}}\mathrm{Bytes}_{\ell,t}^{\mathrm{att}} + \alpha_{q,\ell}^{\mathrm{MAC}}\mathrm{MAC}_{\ell,t}^{\mathrm{att}} \text{ and } \widehat{E}_{q,t}^{\mathrm{att}} := \textstyle\sum_{\ell=1}^{L}\widehat{E}_{q,\ell,t}^{\mathrm{att}}. \tag{18}$$

Thus block $b$ incurs marginal proxy cost $|\mathcal{I}_{b,t}|\,\Delta_{\ell,t}^{\mathrm{tok}}$. Given token-block scores $s_{\ell,b,t}$, we apply the global price $\lambda$ to obtain relaxed gates:

$$\tilde{s}_{\ell,b,t} = s_{\ell,b,t} - \lambda\,|\mathcal{I}_{b,t}|\,\Delta_{\ell,t}^{\mathrm{tok}}, \qquad g_{\ell,b,t} = \Big[\sigma\Big(\frac{\tilde{s}_{\ell,b,t} + \log\varphi_{\ell,b,t} - \log(1-\varphi_{\ell,b,t})}{\mathcal{T}}\Big)(\zeta-\varrho) + \varrho\Big]_{[0,1]}. \tag{19}$$

Here $\varphi_{\ell,b,t}\sim\mathcal{U}(0,1)$ (uniform$(0,1)$), $\mathcal{T}>0$ is a temperature, $\varrho$ and $\zeta$ are stretch parameters.

**FFN gating.** To determine energy-feasible FFN execution at stage $t$, we first allocate the residual budget to FFN computation as follows:

$$E_t^{\mathrm{ffn}} = \big[E_t^{\mathrm{rem}} - \widehat{E}_{q,t}^{\mathrm{att}}\big]_+, \tag{20}$$

which constrains the FFN energy that scales with the effective width. Using $\widehat{E}^{\mathrm{ffn}}_{q,\ell,t} = \beta^{\mathrm{HBM}}_{q,\ell}\,\mathrm{Bytes}^{\mathrm{ffn}}_{\ell,t} + \beta^{\mathrm{MAC}}_{q,\ell}\,\mathrm{MAC}^{\mathrm{ffn}}_{\ell,t}$, the per-block FFN marginal cost with block size $G$ is $e^{\mathrm{ffn}}_{\ell,t} := \frac{\partial \widehat{E}^{\mathrm{ffn}}_{q,\ell,t}}{\partial \bar{w}_{\ell,t}} \cdot G = \left(\beta^{\mathrm{HBM}}_{q,\ell}\, c_{1,\ell} + \beta^{\mathrm{MAC}}_{q,\ell}\, v_{1,\ell}\right) \bar{\kappa}_{\ell,t}\, G$, and we set $e_{\ell,j,t} = e^{\mathrm{ffn}}_{\ell,t}$ for all $j \in \mathcal{J}_\ell$. We price channel-block scores $z_{\ell,j,t}$ via $\lambda$ to obtain priced scores:

$$\tilde{z}_{\ell,j,t} = \sigma(z_{\ell,j,t} - \lambda\, e_{\ell,j,t}) \in (0,1). \tag{21}$$

Because all FFN channel blocks at stage $t$ draw from the same residual FFN energy budget $E^{\mathrm{ffn}}_t$, their execution decisions must be coordinated jointly across layers. Accordingly, we stack $\tilde{z}_t := (\tilde{z}_{\ell,j,t})_{\ell,j}$ and $e_t := (e_{\ell,j,t})_{\ell,j}$, and define the relaxed FFN execution gates at stage $t$ as $m_t := (m_{\ell,j,t})_{\ell,j}$, obtained by solving:

$$m_t = \arg\min_{0\le m\le 1,\, e_t^\top m \le E^{\mathrm{ffn}}_t} \tfrac{1}{2}\|m - \tilde{z}_t\|_2^2. \tag{22}$$

**Training with a single-contract Lagrangian.** Let $\pi_\phi \in \mathcal{Q}$ denote the execution policy parameterized by selector parameters $\phi$, with the pretrained backbone $\theta$ fixed. Under teacher forcing, we train $\phi$ under a single per-sequence energy contract using a global dual price $\lambda \ge 0$:

$$\min_{\phi}\max_{\lambda\ge 0}\ \mathbb{E}_{(x,y)\sim\mathcal{D}}\Big[-\sum_{t=1}^{T}\log p_{\theta,\pi_\phi}(y_t \mid x, y_{<t}) + \lambda\Big(\widehat{E}_q(x,y;\pi_\phi) - E_{\mathrm{bud}}(x,T)\Big)\Big]. \tag{23}$$

The global dual price is updated by doing stochastic ascent:

$$\lambda \leftarrow \Big[\lambda + \eta_\lambda\big(\widehat{E}_q(x,y;\pi_\phi) - E_{\mathrm{bud}}(x,T)\big)\Big]_+, \tag{24}$$

where $\eta_\lambda > 0$ is the dual step size.

### 3.5 Inference-Time Execution

At inference time, EOP-LLM converts the learned, energy-priced execution decisions into concrete token and FFN executions that strictly satisfy a per-sequence energy budget. Given a request $(x, E_{\mathrm{bud}}(x,T))$, execution proceeds stage by stage while maintaining a remaining proxy budget $E^{\mathrm{rem}}_t$ via the recursion in Eq. 16.

At each stage $t$, we first check whether any forward progress is feasible under the energy proxy by computing the minimum execution energy required at that stage. Let $b^\star(t) := \lceil n_t/B \rceil$ denote the token block containing the current position $n_t$. For token retention, the minimum token-block set is $\mathcal{B}^{\min}_{\ell,t} := \{b^\star(t)\}$ for all $\ell \in [L]$, with the corresponding minimum attention energy $\widehat{E}^{\mathrm{att,min}}_{q,t} := \widehat{E}^{\mathrm{att}}_{q,t}(\{\mathcal{B}^{\min}_{\ell,t}\}_{\ell=1}^L)$. We also enforce a minimum FFN width of one channel block per layer, which defines a minimum FFN energy $\widehat{E}^{\mathrm{ffn,min}}_{q,t} := \sum_{\ell=1}^L e^{\mathrm{ffn}}_{\ell,t}$. If $E^{\mathrm{rem}}_t < \widehat{E}^{\mathrm{att,min}}_{q,t} + \widehat{E}^{\mathrm{ffn,min}}_{q,t}$, the budget is insufficient, and execution is terminated.

**Token hardening.** To convert relaxed token decisions into a concrete, energy-feasible token execution, we first compute the attention energy planned by the relaxed token gates, denoted by $E^{\mathrm{att,rlx}}_{q,t}$, using Eq. 18 together with the relaxed retained lengths in Eq. 5. Based on the token-block importance scores $\{s_{\ell,b,t}\}$, we then harden the token gates by retaining higher-scored token blocks subject to $\widehat{E}^{\mathrm{att}}_{q,t}(\{\widehat{\mathcal{B}}_{\ell,t}\}_{\ell=1}^L) \le E^{\mathrm{att,rlx}}_{q,t}$, $b^\star(t) \in \widehat{\mathcal{B}}_{\ell,t}$, and $\widehat{\mathcal{B}}_{\ell,t} \subseteq \widehat{\mathcal{B}}_{\ell-1,t}$, with $\widehat{\mathcal{B}}_{0,t} := \mathcal{B}_t$. The resulting binary token masks are $\widehat{g}_{\ell,b,t} := \mathbf{1}\{b \in \widehat{\mathcal{B}}_{\ell,t}\}$.

**FFN hardening.** Given the hardened token sets, we compute the resulting proxy attention energy $\widehat{E}^{\mathrm{att}}_{q,t} := \widehat{E}^{\mathrm{att}}_{q,t}(\{\widehat{\mathcal{B}}_{\ell,t}\}_{\ell=1}^L)$ and allocate the residual stage budget to FFN execution: $E^{\mathrm{ffn}}_t = \big[E^{\mathrm{rem}}_t - \widehat{E}^{\mathrm{att}}_{q,t}\big]_+$. We then harden FFN channel blocks by constructing a binary mask $\widehat{m}_t \in \{0,1\}^{d_m}$, $d_m := \sum_{\ell=1}^L |\mathcal{J}_\ell|$, based on the importance scores $\{z_{\ell,j,t}\}$. Channel blocks are retained in decreasing order of these scores until the FFN proxy budget $e_t^\top \widehat{m}_t \le E^{\mathrm{ffn}}_t$ is satisfied. The hardened FFN masks are given by $\widehat{m}_{\ell,j,t} := \mathbf{1}\{j \in \widehat{\mathcal{J}}_{\ell,t}\}$, where $\widehat{\mathcal{J}}_{\ell,t}$ denotes the selected channel-block indices at layer $\ell$.

**Execution sets.** Based on these binary masks, the token positions and FFN channels executed at each layer and stage are given by:

$$\mathcal{S}_{\ell,t} = \bigcup_{b:\ \widehat{g}_{\ell,b,t}=1} \mathcal{I}_{b,t} \subseteq \{1,\dots,n_t\}, \qquad \mathcal{C}_{\ell,t} = \bigcup_{j:\ \widehat{m}_{\ell,j,t}=1} \mathcal{K}_{\ell,j} \subseteq \{1,\dots,d^{(\ell)}_{\mathrm{ff}}\}, \tag{25}$$

with the resulting executed sizes $n^{\mathrm{keep}}_{\ell,t} = |\mathcal{S}_{\ell,t}|$ and $w^{\mathrm{keep}}_{\ell,t} = |\mathcal{C}_{\ell,t}|$.

**Remark.** EOP-LLM operates on standard decoder-only Transformer components, namely attention and FFN blocks. Since it does not rely on architecture-specific assumptions, it can be directly applied across different decoder-only LLM families via the same calibration procedure.

## 4 Experiments

**Models and evaluation.** We evaluate EOP-LLM on three pretrained decoder-only transformer models: LLaMA 3.2-1B and LLaMA 3.2-3B (Meta, 2024b), and LLaMA 3.1-8B (Meta, 2024a). All backbone model parameters are kept *frozen* throughout training. To encourage generalizable pruning behavior, we train the pruning selectors and coordination modules of EOP-LLM on a 5M-token mixed public corpus comprising 70% Falcon-RefinedWeb (Penedo et al., 2023), 15% PG-

| Model | Method | $\Psi = 0.5$ | | $\Psi = 0.6$ | | $\Psi = 0.7$ | | $\Psi = 0.8$ | | $\Psi = 1.0$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **PPL** | $\Delta E\%$ | **PPL** | $\Delta E\%$ | **PPL** | $\Delta E\%$ | **PPL** | $\Delta E\%$ | **PPL** | $\Delta E\%$ |
| LLaMA 3.2-1B | Dense | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | 16.35 | 0 |
| | DCP | 550.74 | -1.23 | 220.43 | -1.02 | 32.33 | -0.82 | 22.55 | -0.61 | 16.35 | 0 |
| | PP | 133.51 | -1.58 | 85.71 | -1.35 | 26.95 | -0.77 | 19.85 | -0.58 | 16.35 | 0 |
| | Ours (token-only) | 510.32 | 0.39 | 203.43 | 0.47 | 30.14 | 0.34 | 20.55 | 0.32 | 16.35 | 0 |
| | Ours (FFN-only) | 121.76 | 0.41 | 73.44 | 0.52 | 24.01 | 0.49 | 18.49 | 0.51 | 16.35 | 0 |
| | **Ours** | **38.07** | **0.22** | **20.29** | **0.23** | **17.75** | **0.19** | **16.93** | **0.17** | 16.35 | 0 |
| LLaMA 3.2-3B | Dense | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | 13.12 | 0 |
| | DCP | 292.21 | -1.21 | 171.11 | -1.23 | 26.32 | -1.02 | 17.61 | -0.71 | 13.12 | 0 |
| | PP | 91.78 | -1.47 | 53.45 | -1.15 | 22.31 | -0.97 | 15.12 | -1.38 | 13.12 | 0 |
| | Ours (token-only) | 270.76 | 0.49 | 157.91 | 0.45 | 24.54 | 0.32 | 16.05 | 0.35 | 13.12 | 0 |
| | Ours (FFN-only) | 87.48 | 0.42 | 49.8 | 0.48 | 20.18 | 0.24 | 14.78 | 0.31 | 13.12 | 0 |
| | **Ours** | **21.26** | **0.21** | **15.75** | **0.22** | **14.11** | **0.17** | **13.22** | **0.16** | 13.12 | 0 |
| LLaMA 3.1-8B | Dense | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | 10.73 | 0 |
| | DCP | 266.92 | -1.21 | 148.62 | -1.13 | 22.78 | -0.92 | 14.41 | -0.66 | 10.73 | 0 |
| | PP | 75.74 | -1.47 | 47.95 | -1.25 | 18.58 | -0.87 | 13.05 | -0.98 | 10.73 | 0 |
| | Ours (token-only) | 247.33 | 0.49 | 137.16 | 0.46 | 21.24 | 0.33 | 13.14 | 0.34 | 10.73 | 0 |
| | Ours (FFN-only) | 69.82 | 0.42 | 43.12 | 0.35 | 17.33 | 0.17 | 12.73 | 0.24 | 10.73 | 0 |
| | **Ours** | **19.42** | **0.19** | **13.68** | **0.16** | **12.21** | **0.11** | **10.82** | **0.09** | 10.73 | 0 |

Table 1: Average PPL over 3 trials on WikiText-2 (raw) at target ratios $\Psi$.

| Model | Method | $\Psi$ | $\Delta E\%$ | BoolQ | PIQA | HellaSwag | WinoG | ARC-c | ARC-e | OBQA | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LLaMA 3.2-1B | Dense | 1.00 | 0.00 | 57.77 | 72.96 | 55.54 | 54.38 | 31.44 | 49.63 | 29.80 | 50.22 |
| | DCP | 0.60 | -1.02 | 43.46 | 48.48 | 39.93 | 41.12 | 19.06 | 29.26 | 18.60 | 34.27 |
| | PP | 0.60 | -1.35 | 51.96 | 65.94 | 46.81 | 48.46 | 24.75 | 40.37 | 23.40 | 43.10 |
| | Ours (token-only) | 0.60 | 0.47 | 45.41 | 50.87 | 40.43 | 42.62 | 20.74 | 31.48 | 19.60 | 35.88 |
| | Ours (FFN-only) | 0.60 | 0.52 | 52.45 | 67.57 | 47.92 | 49.49 | 25.08 | 41.48 | 23.80 | 43.97 |
| | **Ours** | 0.60 | **0.21** | **55.93** | **70.13** | **52.47** | **54.14** | **29.43** | **47.04** | **27.60** | **48.11** |
| | DCP | 0.50 | -1.23 | 38.72 | 44.18 | 32.33 | 38.75 | 16.72 | 27.04 | 16.20 | 30.56 |
| | PP | 0.50 | -1.58 | 47.89 | 60.99 | 38.52 | 47.43 | 22.07 | 34.07 | 20.20 | 38.74 |
| | Ours (token-only) | 0.50 | 0.48 | 38.96 | 45.81 | 32.81 | 39.07 | 17.06 | 27.78 | 16.40 | 31.13 |
| | Ours (FFN-only) | 0.50 | 0.34 | 49.36 | 63.49 | 39.45 | 47.91 | 22.41 | 35.56 | 21.40 | 39.94 |
| | **Ours** | 0.50 | **0.22** | **54.16** | **69.48** | **49.11** | **53.35** | **27.42** | **44.81** | **24.80** | **46.16** |

Table 2: Average commonsense accuracy % over 3 trials on seven benchmarks for LLaMA 3.2–1B.

19 (Rae et al., 2019), 10% WikiText-103-raw (Merity et al., 2016), and 5% StackExchange (Stack Exchange, Inc., 2024). Following Dynamic Context Pruning (DCP) (Anagnostidis et al., 2023) and PP (Le et al., 2025), we evaluate perplexity (PPL) on the WikiText-2-raw (Merity et al., 2016) text generation task under teacher forcing with a fixed sequence length of 1024 tokens using the model's native tokenizer. In addition, we evaluate accuracy on standard commonsense reasoning tasks, including BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-c and ARC-e (Clark et al., 2018), and OpenBookQA (Mihaylov et al., 2018), with all sequences in each batch set to the length of the longest sample. All our experiments are conducted on a GPU server with 8× H200. Implementation details and additional experimental settings are provided in Appendix D.1.

**Energy model calibration.** Prior to selector training, we calibrate the device-specific energy proxy on the target hardware for each evaluated backbone (LLaMA 3.2-1B/3B and LLaMA 3.1-8B). Calibration is performed using controlled sweeps of real forward passes that independently vary token retention and FFN width, while holding all other execution dimensions fixed. Measured on-device energy is used to fit the proxy coefficients via non-negative ridge regression. The implementation details and validation results are provided in Appendix D.2.

| Model | Method | Ψ | $\Delta E\%$ | BoolQ | PIQA | HellaSwag | WinoG | ARC-c | ARC-e | OBQA | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LLaMA 3.2-3B | Dense | 1.00 | 0.00 | 68.81 | 75.19 | 65.57 | 63.69 | 39.13 | 58.15 | 36.80 | 58.19 |
| | DCP | 0.60 | -1.23 | 53.82 | 60.23 | 49.17 | 48.15 | 23.08 | 38.15 | 24.20 | 42.40 |
| | PP | 0.60 | -1.15 | 61.16 | 71.93 | 59.81 | 58.25 | 31.44 | 50.37 | 30.60 | 51.94 |
| | Ours (token-only) | 0.60 | 0.45 | 54.83 | 61.70 | 50.33 | 49.41 | 24.48 | 39.26 | 24.80 | 43.54 |
| | Ours (FFN-only) | 0.60 | 0.48 | 62.91 | 72.03 | 60.67 | 59.75 | 32.78 | 51.85 | 31.80 | 53.11 |
| | **Ours** | 0.60 | **0.21** | **66.36** | **74.21** | **62.72** | **63.06** | **35.45** | **55.56** | **34.20** | **55.94** |
| | DCP | 0.50 | -1.21 | 49.97 | 52.71 | 41.14 | 45.54 | 21.41 | 34.22 | 20.80 | 37.97 |
| | PP | 0.50 | -1.47 | 58.13 | 66.76 | 53.34 | 55.41 | 27.09 | 45.56 | 27.20 | 47.64 |
| | Ours (token-only) | 0.50 | 0.49 | 51.68 | 53.68 | 42.49 | 46.25 | 21.75 | 34.97 | 21.20 | 38.86 |
| | Ours (FFN-only) | 0.50 | 0.42 | 59.76 | 67.74 | 55.32 | 57.22 | 27.76 | 46.67 | 28.20 | 48.95 |
| | **Ours** | 0.50 | **0.23** | **63.12** | **73.01** | **59.87** | **61.72** | **30.77** | **52.22** | **30.20** | **52.99** |

Table 3: Average commonsense accuracy % over 3 trials on seven benchmarks for LLaMA 3.2–3B.

| Model | Method | Ψ | $\Delta E\%$ | BoolQ | PIQA | HellaSwag | WinoG | ARC-c | ARC-e | OBQA | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LLaMA 3.1-8B | Dense | 1.00 | 0.00 | 80.28 | 78.24 | 76.18 | 71.19 | 46.49 | 62.59 | 43.20 | 65.45 |
| | DCP | 0.60 | -1.13 | 62.75 | 64.64 | 55.27 | 52.72 | 27.09 | 43.33 | 28.60 | 47.77 |
| | PP | 0.60 | -1.25 | 72.72 | 73.88 | 66.12 | 65.11 | 34.11 | 54.07 | 34.40 | 57.20 |
| | Ours (token-only) | 0.60 | 0.46 | 63.33 | 65.78 | 56.28 | 53.12 | 27.76 | 44.44 | 28.80 | 48.50 |
| | Ours (FFN-only) | 0.60 | 0.35 | 74.98 | 74.43 | 66.77 | 65.51 | 35.79 | 54.81 | 34.60 | 58.13 |
| | **Ours** | 0.60 | **0.19** | **78.29** | **77.86** | **72.97** | **69.06** | **42.14** | **59.26** | **40.20** | **62.83** |
| | DCP | 0.50 | -1.21 | 55.47 | 56.91 | 43.38 | 47.36 | 23.08 | 35.19 | 23.40 | 40.68 |
| | PP | 0.50 | -1.47 | 68.53 | 70.13 | 61.12 | 59.98 | 30.43 | 50.37 | 30.60 | 53.02 |
| | Ours (token-only) | 0.50 | 0.49 | 56.36 | 57.13 | 44.38 | 47.75 | 23.41 | 37.78 | 23.80 | 41.52 |
| | Ours (FFN-only) | 0.50 | 0.42 | 69.82 | 71.44 | 62.06 | 60.22 | 30.77 | 50.74 | 31.20 | 53.75 |
| | **Ours** | 0.50 | **0.22** | **74.34** | **75.14** | **68.38** | **65.98** | **38.13** | **56.67** | **36.80** | **59.35** |

Table 4: Average commonsense accuracy % over 3 trials on seven benchmarks for LLaMA 3.1–8B

**Baselines.** Given on-device serving constraints, where each inference request must satisfy a fixed per-sequence energy budget $E_{\text{bud}}(x, T)$, we compare EOP-LLM against two representative inference-time dynamic pruning methods: DCP (Anagnostidis et al., 2023), a token-only dynamic pruning method that reduces the effective attention context by selecting a subset of prefix tokens while keeping the model width unchanged, and PP (Le et al., 2025), a width-only dynamic pruning method that prunes attention and FFN channels via probe-based importance estimation while retaining all tokens. To ensure a fair comparison, we use the official codes of DCP and PP and follow their default configurations. Since neither method directly enforces a per-sequence energy budget during inference, we perform a sweep over their pruning hyperparameters (e.g., retention or sparsity levels) and, for each target budget, select the setting whose measured energy is closest to the target. Note that we do not include offline fixed compression methods (e.g., GreenLLM), as these methods produce fixed models and do not support enforcing explicit per-sequence energy budgets or adapting execution at inference time. In addition, we include dense (no pruning) and two ablated variants of EOP-LLM to isolate the contribution of each pruning component: (i) token-only EOP, which disables FFN width selection and performs only energy-priced token retention, and (ii) FFN-only EOP, which retains all tokens and performs only energy-priced FFN channel-block selection.

**Main results.** We evaluate all methods under explicit per-sequence energy budgets parameterized by a target ratio $\Psi \in (0, 1]$, where $\Psi$ specifies the desired budget relative to dense execution. For each method and target $\Psi$, we report both quality and energy-budget adherence, defined as $\Delta E\% := 100 \cdot \frac{E_{\text{bud}} - E_{\text{meas}}}{E_{\text{bud}}}$, where $E_{\text{meas}}$ denotes the measured end-to-end inference energy. Under this definition, $\Delta E\% < 0$ indicates energy overuse, while $\Delta E\% > 0$ indicates energy underuse.

We first evaluate zero-shot text generation quality under energy targets ranging from moderate to aggressive ($\Psi \in \{0.8, 0.7, 0.6, 0.5\}$). Table 1 shows that EOP-LLM consistently achieves lower perplexity than DCP and PP at the same target $\Psi$, while maintaining positive energy-budget adherence. At the most aggressive setting $\Psi = 0.5$, EOP-LLM substantially outperforms the strongest baseline (PP), reducing perplexity from $133.51 \rightarrow 38.07$ (1B), $91.78 \rightarrow 21.26$ (3B), and $75.74 \rightarrow 19.42$ (8B), while remaining within budget with $\Delta E\% = 0.22/0.21/0.19$. Note that Enforcing strict budget adherence ($\Delta E\% \geq 0$) for PP and DCP would require more aggressive pruning and would further degrade their quality; accordingly, these baselines are evaluated at their closest-energy settings, allowing small negative $\Delta E\%$, which makes the comparison conservative in their favor. A similar performance pattern is observed on commonsense reasoning. As shown in Table 4 and Tables 2 and 3 in Appendix D.3, EOP-LLM consistently outperforms PP under the same energy budgets across all seven benchmarks. At $\Psi = 0.5$, EOP-LLM achieves average accuracy gains of $7.42\%$, $5.35\%$, and $6.33\%$ on the 1B, 3B, and 8B models, respectively, while remaining within the target budget.

The consistent gains observed across all evaluated settings are not surprising, given that EOP-LLM explicitly aligns pruning decisions with device energy. DCP (token-only) and PP (width-only) rely on importance signals without accounting for the layer-wise energy cost of their decisions, and thus can misallocate computation under a fixed budget. In contrast, EOP-LLM employs a device-calibrated, layer-aware energy proxy to jointly decide token retention and FFN width, allocating computation to the most impactful layers while satisfying the energy constraint.

**Effects of the proposed components.** We assess the contributions of the core components of EOP-LLM by comparing the full method against token-only and FFN-only variants under the same energy budgets. Across both text generation and commonsense reasoning (Tables 1–3), the full EOP-LLM achieves the best quality under the same energy budgets. At the tight budget $\Psi = 0.5$, the full EOP-LLM reduces perplexity from $510.32 \rightarrow 38.07$ (token-only) and $121.76 \rightarrow 38.07$ (FFN-only), and improves average commonsense accuracy by up to $17.83\%$ and $6.22\%$, respectively. These results confirm that jointly allocating computation between token retention and FFN width is more effective than token-only or FFN-only pruning for maximizing quality under explicit energy constraints.

We further analyze the impact of token block size $B$ and channel block size $G$ under a fixed energy budget ($\Psi = 0.6$). This experiment aims to examine how execution granularity affects both budget controllability and downstream reasoning performance.

| $\Psi$ | $B$ | $G$ | $\Delta E$ (%) | WinoG | ARC-c | BoolQ |
|---|---|---|---|---|---|---|
| 0.6 | 8 | 64 | 0.22 | 63.06 | 35.45 | 66.36 |
| 0.6 | 64 | 64 | 0.31 | 61.17 | 34.11 | 63.67 |
| 0.6 | 128 | 64 | 0.43 | 57.14 | 30.76 | 59.08 |
| 0.6 | 8 | 128 | 0.28 | 62.98 | 35.12 | 65.75 |
| 0.6 | 8 | 256 | 0.34 | 61.96 | 32.78 | 65.05 |

Table 5: Block-size ablation of EOP-LLM at $\Psi = 0.6$ on LLaMA-3.2-3B. Average accuracy (%) over 3 trials.

As shown in Table 5, increasing either $B$ or $G$ consistently increases $\Delta E$, indicating weaker budget controllability under coarser execution granularity. The reason is that larger token or channel blocks increase the minimum energy adjustment step, so each keep/drop decision changes the realized energy in larger increments, making precise budget matching more difficult. At the same time, performance degrades as $B$ or $G$ increases, because coarser blocks force more computational units to share a single execution decision, reducing the selector's ability to preserve fine-grained important structures.

Beyond the main evaluations, Appendix D.3.2–D.3.6 reports additional analyses on component overhead, prefill and decode efficiency, learned execution behavior, and energy budget adherence.

## 5 Conclusion

We introduce EOP-LLM, an energy-oriented dynamic pruning framework for inference under explicit energy budget constraints. EOP-LLM combines a device-calibrated energy proxy with lightweight token and FFN selectors, coordinated through a global dual variable, to adapt computation on a per-sequence basis while preserving model quality. Extensive experiments demonstrate that EOP-LLM consistently outperforms state-of-the-art dynamic pruning methods under matched energy budgets, without violating the energy budget constraints. Looking ahead, we plan to extend EOP-LLM to mixture-of-experts architectures by augmenting the energy proxy to capture expert routing and sparse execution. More broadly, EOP-LLM also provides a step toward applying LLMs to LEO satellite systems (Huang and Shu, 2024, 2025), where energy constraints are fundamental.

## Limitations

While EOP-LLM provides a promising approach for enabling LLM inference under explicit per-sequence energy budgets, several limitations warrant consideration.

First, EOP-LLM relies on a device-calibrated energy proxy and auxiliary selectors that are calibrated and trained on the target hardware. While this overhead is substantially smaller than fine-tuning the backbone model, calibration sweeps may need to be repeated when kernels or drivers change. Furthermore, to reduce the risk of proxy underestimation, we adopt a conservative guard margin, but it can introduce small positive slack and slightly underutilize the available budget.

For evaluation, we use pretrained decoder-only LLaMA backbones (LLaMA 3.2 1B/3B and LLaMA 3.1 8B), which are widely used publicly available models. Most experiments are conducted on their base versions rather than instruction-tuned variants, following prior work (Guo et al., 2025; Katz et al., 2025). This design choice isolates the effects of energy-constrained execution from confounding factors such as model-specific prompt templates and system prompts, enabling a more controlled comparison of model quality under different energy budgets. While EOP-LLM consistently outperforms state-of-the-art dynamic pruning baselines under given energy budgets, we acknowledge that there may still be rooms to improve the EOP-LLM in specialized LLMs by exploiting additional conditions of the language models.

We evaluate EOP-LLM on zero-shot text generation and commonsense reasoning benchmarks drawn from prior work (Le et al., 2025; Katz et al., 2025). Future work will extend EOP-LLM to dialogue-based generation settings (e.g., multi-turn conversational or instruction-following chat models), as well as to sequence-to-sequence tasks such as translation and text summarization.

## Ethical Considerations

This work studies energy-constrained inference for large language models using publicly available pretrained models and standard benchmarks. It does not involve human subjects, personal data, or user interaction. The proposed method modifies inference-time execution without introducing new model capabilities. As with any deployment of language models, downstream applications should follow established best practices for responsible and safe use.


## Acknowledgments

This work is supported in part by the United States National Science Foundation (NSF) under grants CNS-2308761 and CNS-2006998. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the author(s) and do not necessarily reflect the views of NSF.


## References


Sotiris Anagnostidis, Dario Pavllo, Luca Biggio, Lorenzo Noci, Aurelien Lucchi, and Thomas Hofmann. 2023. Dynamic context pruning for efficient and interpretable autoregressive transformers. *Advances in Neural Information Processing Systems*, 36:65202–65223.

Mauricio Fadel Argerich and Marta Patiño-Martínez. 2024. Measuring and improving the energy efficiency of large language models inference. *IEEE Access*, 12:80194–80207.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, and 1 others. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, 05, pages 7432–7439.

Francisco Caravaca, Ángel Cuevas, and Rubén Cuevas. 2025. From prompts to power: Measuring the energy footprint of llm inference. *arXiv preprint arXiv:2511.05597*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Quentin Colmant and Federico Marcuzzi. 2023. pyrapl: A python wrapper for intel rapl energy measurements. https://pypi.org/project/pyRAPL/.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359.

DESNZ. 2025. Digest of united kingdom energy statistics (dukes) 2025 — chapter 5: Electricity. https://assets.publishing.service.gov.uk/media/688a\2865478525675739051/DUKES_2025_Chapter_5.pdf.

Jingzhi Fang, Yanyan Shen, Yue Wang, and Lei Chen. 2025. Improving the end-to-end efficiency of offline inference for multi-llm applications based on sampling and simulation. *arXiv preprint arXiv:2503.16893*.

Jared Fernandez, Clara Na, Vashisth Tiwari, Yonatan Bisk, Sasha Luccioni, and Emma Strubell. 2025. Energy considerations of large language model inference and efficiency optimizations. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32556–32569, Vienna, Austria. Association for Computational Linguistics.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Qichen Fu, Minsik Cho, Thomas Merth, Sachin Mehta, Mohammad Rastegari, and Mahyar Najibi. 2024. Lazyllm: Dynamic token pruning for efficient long context llm inference. *arXiv preprint arXiv:2407.14057*.

Jialong Guo, Xinghao Chen, Yehui Tang, and Yunhe Wang. 2025. Slimllm: Accurate structured pruning for large language models. *arXiv preprint arXiv:2505.22689*.

Guan Huang and Tao Shu. 2024. A global-local probsparse self-attention transformer for leo satellite orbit prediction. In *2024 International Conference on Machine Learning and Applications (ICMLA)*, pages 91–98.

Guan Huang and Tao Shu. 2025. Federated oriented learning: A practical one-shot personalized federated learning framework. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 25762–25790. PMLR.

Wei Huang, Haotong Qin, Yangdong Liu, Yawei Li, Qinshuo Liu, Xianglong Liu, Luca Benini, Michele Magno, Shiming Zhang, and Xiaojuan Qi. 2024. Slim-llm: Salience-driven mixed-precision quantization for large language models. *arXiv preprint arXiv:2405.14917*.

Erik Johannes Husom, Arda Goknil, Merve Astekin, Lwin Khin Shar, Andre Kåsen, Sagar Sen, Benedikt Andreas Mithassel, and Ahmet Soylu. 2025. Sustainable llm inference for edge ai: Evaluating quantized llms for energy efficiency, output accuracy, and inference latency. *arXiv preprint arXiv:2504.03360*.

Erik Johannes Husom, Arda Goknil, Lwin Khin Shar, and Sagar Sen. 2024. The price of prompting: Profiling energy use in large language models inference. *arXiv preprint arXiv:2407.16893*.

Luke James. 2025. Tesla targets ai data centers with massive megapack batteries as grid-strain fears grow — says $50b/gw for a 2-hour system over a 20-year lifetime is 'outsized value'. https://www.tomshardware.com/tech-\industry/tesla-targets-ai-\data-centers-with-\megapack-as-grid-strain-fears-grow.

Nidhal Jegham, Marwan Abdelatti, Chan Young Koh, Lassad Elmoubarki, and Abdeltawab Hendawi. 2025. How hungry is ai? benchmarking energy, water, and carbon footprint of llm inference. *arXiv preprint arXiv:2505.09598*.

Yunho Jin, Gu-Yeon Wei, and David Brooks. 2025. The energy cost of reasoning: Analyzing energy usage in llms with test-time compute. *arXiv preprint arXiv:2505.14733*.

Shahar Katz, Liran Ringel, Yaniv Romano, and Lior Wolf. 2025. Segment-based attention masking for gpts. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 19308–19322.

Laila Kearney. 2025. Google agrees to curb power use for ai data centers to ease strain on us grid when demand surges. https://www.reuters.com/sustainability/boards\-policy-regulation/\google-agrees-curb-power-\use-ai-data-centers-ease\-strain-us-grid-when-demand-2025-08-04/.

Rajeev Kumar, Kumar Ishan, Harishankar Kumar, and Abhinandan Singla. 2025. Llm-powered knowledge graphs for enterprise intelligence and analytics. *arXiv preprint arXiv:2503.07993*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.

Qi Le, Enmao Diao, Ziyan Wang, Xinran Wang, Jie Ding, Li Yang, and Ali Anwar. 2025. Probe pruning: Accelerating LLMs through dynamic pruning via model-probing. In *The Thirteenth International Conference on Learning Representations*.

Rebecca Leppert. 2025. What we know about energy use at u.s. data centers amid the ai boom. https://www.pewresearch.org/short-reads/2025/10/24/what\-we-know-about-energy-use-\at-us-data-centers-amid-the-ai-boom/.

Guannan Liang and Qianqian Tong. 2025. Llm-powered ai agent systems and their applications in industry. *arXiv preprint arXiv:2505.16120*.

Yijin Liu, Fandong Meng, and Jie Zhou. 2024. Accelerating inference in large language models with a unified layer skipping strategy. *arXiv preprint arXiv:2404.06954*.

Lingkun Long, Rubing Yang, Yushi Huang, Desheng Hui, Ao Zhou, and Jianlei Yang. 2025. Sliminfer: Accelerating long-context llm inference via dynamic token pruning. *arXiv preprint arXiv:2508.06447*.

Xuan Luo, Weizhi Wang, and Xifeng Yan. 2025. Diffskip: Differential layer skipping in large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 7221–7231.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Meta. 2024a. Introducing llama 3.1. Meta AI Blog. https://ai.meta.com/blog/meta-llama-3-1/.

Meta. 2024b. Llama 3.2: Revolutionizing edge ai and vision. Meta AI Blog. https://ai.meta.com/blog/llama-3-2-connect-2024-vision\-edge-mobile-devices/.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.

Saurav Muralidharan, Sharath Turuvekere Sreenivas, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. 2024. Compact language models via pruning and knowledge distillation. *Advances in Neural Information Processing Systems*, 37:41076–41102.

Chenxu Niu, Wei Zhang, Jie Li, Yongjian Zhao, Tongyang Wang, Xi Wang, and Yong Chen. 2025. Tokenpowerbench: Benchmarking the power consumption of llm inference. *arXiv preprint arXiv:2512.03024*.

NVIDIA. 2024. Pynvml: Python bindings for nvidia management library. https://pypi.org/project/pynvml/.

NVIDIA. 2025. cuBLAS library documentation. NVIDIA Developer Documentation. https://docs.nvidia.com/cuda/cublas/index.html.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The refinedweb dataset for falcon llm: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*.

Liu Qianli, Hong Zicong, Chen Fahao, Li Peng, and Guo Song. 2025. Mell: Memory-efficient large language model serving via multi-gpu kv cache management. *arXiv preprint arXiv:2501.06709*.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. 2019. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Olivia Sidoti and Colleen McClain. 2025. 34% of u.s. adults have used chatgpt, about double the share in 2023. https://www.pewresearch.org/short-reads/2025/06/25/34-of-us-adults-\have-used-chatgpt-\about-double-the-share-in-2023/.

Stack Exchange, Inc. 2024. Stack exchange data dump. https://archive.org/details/stackexchange. Public archive of Stack Exchange user contributions (Creative Commons BY-SA).

Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. 2025. Dynamollm: Designing llm inference clusters for performance and energy efficiency. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1348–1362. IEEE.

Yao Tao, Yehui Tang, Yun Wang, Mingjian Zhu, Hailin Hu, and Yunhe Wang. 2025. Saliency-driven dynamic token pruning for large language models. *arXiv preprint arXiv:2504.04514*.

Chunlin Tian, Xinpeng Qin, and Li Li. 2024. Greenllm: Towards efficient large language model via energy-aware pruning. In *2024 IEEE/ACM 32nd International Symposium on Quality of Service (IWQoS)*, pages 1–2. IEEE.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Shashank Verma and Neal Vaidya. 2023. Mastering llm techniques: Inference optimization. NVIDIA Developer Blog.

Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Jiachen Liu, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, and 1 others. 2023. Efficient large language models: A survey. *arXiv preprint arXiv:2312.03863*.

Dan Wang, Boan Liu, Rui Lu, Zhaorui Zhang, and Shuntao Zhu. 2025. Storellm: Energy efficient large language model inference with permanently pre-stored attention matrices. In *Proceedings of the 16th ACM International Conference on Future and Sustainable Energy Systems*, pages 398–406.

Tianhua Xia and Sai Qian Zhang. 2025. Kelle: Co-design kv caching and edram for efficient llm serving in edge computing. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture®*, pages 18–33.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pages 38087–38099. PMLR.

Haoyi Xiong, Jiang Bian, Yuchen Li, Xuhong Li, Mengnan Du, Shuaiqiang Wang, Dawei Yin, and Sumi Helal. 2024. When search engine services meet large language models: Visions and challenges. *IEEE Transactions on Services Computing*, 17(6):4558–4577.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Mingyang Zhang, Hao Chen, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, and Bohan Zhuang. 2024. Loraprune: Structured pruning meets low-rank parameter-efficient fine-tuning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3013–3026.

Anhao Zhao, Fanghua Ye, Yingqi Fan, Junlong Tong, Zhiwei Fei, Hui Su, and Xiaoyu Shen. 2025. Skipgpt: Dynamic layer pruning reinvented with token awareness and module decoupling. *arXiv preprint arXiv:2506.04179*.

Youpeng Zhao, Di Wu, and Jun Wang. 2024. Alisa: Accelerating large language model inference via sparsity-aware kv caching. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 1005–1017. IEEE.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, and 1 others. 2024. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294*.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. 2024. A survey on model compression for large language models. *Transactions of the Association for Computational Linguistics*, 12:1556–1577.

# Appendices

## A Related Work

### A.1 FLOP-oriented LLM Inference Efficiency

FLOP-oriented inference efficiency methods have emerged as a widely adopted approach for improving LLM inference efficiency by reducing computation and latency (Wan et al., 2023; Xiao et al., 2023; Zhu et al., 2024; Zhou et al., 2024; Liu et al., 2024; Fang et al., 2025). Existing FLOP-oriented methods primarily employ two strategies: static model compression and dynamic computation. (1) Static model compression methods reduce inference cost by permanently simplifying the parameterization of a pretrained LLM prior to deployment, resulting in a smaller fixed model executed at inference time. Common techniques include structured or unstructured pruning of weights, channels, or attention heads based on activation statistics or gradient-based importance measures, post-training quantization to reduce numerical precision while preserving accuracy, and knowledge distillation that transfers behavior from a large teacher model to a compact student. Representative examples include GPTQ (Frantar et al., 2022), LLM-Pruner (Ma et al., 2023), LoRAPrune (Zhang et al., 2024), MINITRON (Muralidharan et al., 2024), SliM-LLM (Huang et al., 2024), and SlimLLM (Guo et al., 2025). (2) Dynamic computation methods reduce inference cost by adapting the amount of computation executed for each input at inference time. Prevailing techniques include pruning uninformative tokens or key–value cache entries to shorten the effective attention context during decoding, conditionally skipping transformer layers or attention blocks for inputs assessed to require less computation, and dynamically gating attention or FFN components using signals derived from intermediate hidden states (e.g., activation magnitude, uncertainty, or probe-based importance scores). These mechanisms tailor the executed computation on a per-input basis while preserving the original model structure. Methods in this category include Dynamic Context Pruning (DCP) (Anagnostidis et al., 2023), LazyLLM (Fu et al., 2024), Probe Pruning (Le et al., 2025), SlimInfer (Long et al., 2025), Saliency-based pruning (Tao et al., 2025), SkipGPT (Zhao et al., 2025), and DiffSkip (Luo et al., 2025).

Although FLOP-oriented methods provide effective mechanisms for reducing computation cost and improving latency, they are not well suited to the setting we consider, in which each inference request must satisfy an explicit per-sequence en-

ergy budget on a given device. First, these methods do not explicitly model device-level energy consumption. While they reduce arithmetic operations, actual inference energy also depends on memory traffic, kernel execution behavior, and other system-level factors, causing FLOP reductions to translate inconsistently into energy savings. Second, FLOP-oriented methods do not enforce per-request energy constraints during execution. Their pruning or skipping decisions are instead controlled indirectly through fixed sparsity or retention hyperparameters, which cannot adapt to fluctuating energy headroom or input-dependent variation in computational demand in real-world deployment environments, such as large-scale data centers or resource-constrained edge devices.

### A.2 Energy-related LLM Inference Efficiency

Energy consumption during LLM inference has recently attracted increasing attention, spanning both system-level serving optimizations and model-level efficiency methods, as well as measurement and benchmarking studies that characterize energy behavior under realistic workloads (Fernandez et al., 2025; Husom et al., 2025; Caravaca et al., 2025; Wang et al., 2025; Qianli et al., 2025). Existing energy-related works mainly fall into two groups: energy-aware serving systems and energy measurement and benchmarking. (1) Energy-aware serving systems aim to optimize inference energy at the serving layer by adapting deployment configurations and runtime scheduling decisions, including request routing, device placement, KV cache management, and cluster-level resource control under latency or service-level objectives. Representative examples in this category include StoreLLM (Wang et al., 2025), DynamoLLM (Stojkovic et al., 2025), MELL (Qianli et al., 2025), and Kelle (Xia and Zhang, 2025). (2) Energy measurement and benchmarking focus on empirically characterizing inference energy across models, workloads, hardware platforms, and serving stacks through instrumentation and large-scale measurement. Methods in this category include MELODI (Husom et al., 2024), MIELLM (Argerich and Patiño-Martínez, 2024), ECLLM (Fernandez et al., 2025), FPP (Caravaca et al., 2025), and SLLM (Husom et al., 2025). Beyond these two dominant directions, GreenLLM (Tian et al., 2024) performs offline model compression by using energy measurements to allocate pruning ratios across layers and derive a fixed pruned model for deployment.

While these works provide valuable advances in reducing serving energy and improving our empirical understanding of inference energy behavior, their effectiveness is limited for the setting we consider. Serving-layer methods optimize energy through deployment and scheduling decisions and therefore do not provide a model-internal mechanism to enforce an explicit per-sequence energy budget during execution. GreenLLM rely on fixed compression choices and do not yield a dynamic execution policy capable of enforcing per-input energy constraints at inference time. Measurement and benchmarking studies, while essential for grounding energy claims, primarily diagnose energy behavior rather than providing mechanisms for budget-constrained inference.

## B Pseudocode for EOP-LLM

Algorithm 1 summarizes the inference-time execution of EOP-LLM. One additional operation that warrants clarification is the projection in Eq. 22, which enforces the shared stage-level FFN energy budget by projecting the priced scores $\tilde{z}_t$ onto the feasible set $\{m \in [0,1]^{d_m} : e_t^\top m \le E_t^{\mathrm{ffn}}\}$. By the KKT conditions, the resulting solution can be written in the closed form:

$$m_t = \left[\tilde{z}_t - \mu_t e_t\right]_{[0,1]}, \tag{26}$$

where $[u]_{[0,1]} := \min\{1, \max\{0, u\}\}$ is applied elementwise and $\mu_t \ge 0$ is chosen so that $e_t^\top m_t = E_t^{\mathrm{ffn}}$ when the budget constraint is active (and $\mu_t = 0$ otherwise).

## C Closed-form proxy coefficients.

We instantiate the count terms in Eq. 9 using tensor shapes of layer $\ell$ and the kernel structure of a decoder-only transformer. Let $d$ be the hidden dimension, and let the attention module use $h_{\mathrm{q}}^{(\ell)}$ query heads and $h_{\mathrm{kv}}^{(\ell)}$ KV heads (with head dimension $d_h := d/h_{\mathrm{q}}^{(\ell)}$). The KV vector dimension per token is

$$d_{\mathrm{kv}}^{(\ell)} := h_{\mathrm{kv}}^{(\ell)}\, d_h.$$

Let $b_{\mathrm{w}}$ be the number of bytes per weight element (e.g., 2 for FP16/BF16) and $b_{\mathrm{kv}}$ be the number of bytes per KV-cache element.

**Attention coefficients.** Recall that we model attention-related memory traffic and arithmetic as $\mathrm{Bytes}_{\ell,t}^{\mathrm{att}} = a_{0,\ell} + a_{1,\ell}\,\bar{n}_{\ell,t}$, $\mathrm{MAC}_{\ell,t}^{\mathrm{att}} = u_{1,\ell}\,\mathbb{1}\{t \ge 1\}\,\bar{n}_{\ell,t} + u_{2,\ell}\,\mathbb{1}\{t = 0\}\,\bar{n}_{\ell,t}^2$. where $\bar{n}_{\ell,t}$ is the retained KV-cache length at layer $\ell$ and stage $t$.

**Algorithm 1** EOP-LLM inference-time coordination (relaxation + hardening)

**Require:** Prompt $x$, budget $E_{\mathrm{bud}}(x,T)$, max steps $T$, backbone $f_\theta$, selectors $\mathsf{TokSel}, \mathsf{FFNSel}$, dual price $\lambda$, proxy params $(\alpha,\beta,\gamma_q)$, block sizes $(B,G)$.
**Ensure:** Generated tokens $\hat{y}$.

$\hat{y} \leftarrow \emptyset$
Initialize remaining proxy budget $E_0^{\mathrm{rem}} \leftarrow E_{\mathrm{bud}}(x,T) - \gamma_q$ (Eq. 16)
**for** $t = 0, 1, \ldots, T$ **do**
  Compute prefix length $n_t$, blocks $\{\mathcal{I}_{b,t}\}_{b\in\mathcal{B}_t}$, and $b^\star(t) = \lceil n_t/B \rceil$ (Sec. 3.2)
  $\mathcal{B}_{\ell,t}^{\min} \leftarrow \{b^\star(t)\}$ for all $\ell \in [L]$ (Sec. 3.5)
  $\widehat{E}_{q,t}^{\mathrm{att,min}} \leftarrow \widehat{E}_{q,t}^{\mathrm{att}}(\{\mathcal{B}_{\ell,t}^{\min}\}_{\ell=1}^{L})$
  $\widehat{E}_{q,t}^{\mathrm{ffn,min}} \leftarrow \sum_{\ell=1}^{L} e_{\ell,t}^{\mathrm{ffn}}$ (Sec. 3.5)
  **if** $E_t^{\mathrm{rem}} < \widehat{E}_{q,t}^{\mathrm{att,min}} + \widehat{E}_{q,t}^{\mathrm{ffn,min}}$ **then**
    **break** (insufficient budget to make progress)
  **end if**
  Get token scores $\{s_{\ell,b,t}\} \leftarrow \mathsf{TokSel}(x, \hat{y}_{<t}, t)$ (Eq. 13)
  Compute $\Delta_{\ell,t}^{\mathrm{tok}}$ and relaxed token gates $\{g_{\ell,b,t}\}$ (Eqs. 17,19)
  Compute relaxed retained lengths $\{\bar{n}_{\ell,t}\}$ (Eq. 5)
  Compute planned attention energy $E_{q,t}^{\mathrm{att,rlx}}$ from $\{\bar{n}_{\ell,t}\}$ (Eqs. 18,9)
  $E_t^{\mathrm{ffn,rel}} \leftarrow \left[E_t^{\mathrm{rem}} - E_{q,t}^{\mathrm{att,rlx}}\right]_+$ (Eq. 20)
  Get channel scores $\{z_{\ell,j,t}\} \leftarrow \mathsf{FFNSel}(x, \hat{y}_{<t}, t)$ (Eq. 15)
  Form costs $e_t = (e_{\ell,j,t})_{\ell,j}$ and priced scores $\tilde{z}_t = (\tilde{z}_{\ell,j,t})_{\ell,j}$ (Eq. 21)
  $m_t \leftarrow \mathsf{Project}(\tilde{z}_t, e_t, E_t^{\mathrm{ffn,rel}})$ (Eq. 22, Alg. 26)
  $\{\widehat{\mathcal{B}}_{\ell,t}\}_{\ell=1}^{L} \leftarrow \mathsf{HardenTok}(\{s_{\ell,b,t}\}, b^\star(t), E_{q,t}^{\mathrm{att,rlx}})$ (Sec. 3.5)
  $E_{q,t}^{\mathrm{att,hard}} \leftarrow \widehat{E}_{q,t}^{\mathrm{att}}(\{\widehat{\mathcal{B}}_{\ell,t}\}_{\ell=1}^{L})$
  $E_t^{\mathrm{ffn,hard}} \leftarrow \left[E_t^{\mathrm{rem}} - E_{q,t}^{\mathrm{att,hard}}\right]_+$
  $\widehat{m}_t \leftarrow \mathsf{HardenFFN}(\{z_{\ell,j,t}\}, e_t, E_t^{\mathrm{ffn,hard}})$ (Sec. 3.5)
  Build $\{\mathcal{S}_{\ell,t}, \mathcal{C}_{\ell,t}\}$ from Eq. 25 and execute one stage to obtain $\hat{y}_t$
  Append $\hat{y}_t$ to $\hat{y}$; if EOS then **break**
  Compute stage proxy energy $\widehat{E}_q^{(t)} = \sum_{\ell=1}^{L} \widehat{E}_{q,\ell,t}(n_t, n_{\ell,t}^{\mathrm{keep}}, w_{\ell,t}^{\mathrm{keep}})$ (Eq. 16)
  $E_{t+1}^{\mathrm{rem}} \leftarrow \left[E_t^{\mathrm{rem}} - \widehat{E}_q^{(t)}\right]_+$ (Eq. 16)
  **if** $E_{t+1}^{\mathrm{rem}} = 0$ **then**
    **break**
  **end if**
**end for**
**return** $\hat{y}$

The coefficient $a_{1,\ell}$ captures the per-token memory traffic incurred by reading the retained KV cache from device memory. For each retained token, one key vector and one value vector of dimension $d_{\mathrm{kv}}^{(\ell)}$ are accessed.

Note that, in some attention implementations, the underlying attention kernel requires KV entries to be stored contiguously in memory (e.g., FlashAttention (Dao et al., 2022)). When token pruning produces a non-contiguous set of retained positions, the resulting KV entries cannot be processed directly by such kernels. In such cases, the retained KV entries are packed into a contiguous buffer prior to attention computation. Depending on the attention kernel and runtime implementation, this packing may be realized either logically, by operating on a compacted view of the KV cache without copying data (in-place), or physically, by explicitly reading and copying the retained KV entries into a new contiguous buffer (read–copy). We model the resulting implementation-dependent memory traffic using a constant $\chi_{\mathrm{kv}}$, with $\chi_{\mathrm{kv}} = 0$ for in-place packing and $\chi_{\mathrm{kv}} = 1$ for read–copy

implementations. Accordingly, we define

$$a_{1,\ell} := 2(1 + \chi_{\mathrm{kv}})\, d_{\mathrm{kv}}^{(\ell)}\, b_{\mathrm{kv}}, \tag{27}$$

where $b_{\mathrm{kv}}$ denotes the number of bytes per KV element.

All attention-side memory traffic that is invariant to the retained KV-cache length is grouped into the constant term

$$a_{0,\ell} := \mathrm{Bytes}_{\ell}^{\mathrm{att,fix}}, \tag{28}$$

where $\mathrm{Bytes}_{\ell}^{\mathrm{att,fix}}$ denotes the attention-side memory traffic at layer $\ell$ that is independent of token retention, including query-side activation reads, attention projection weights, output writes, and fixed kernel or metadata overhead. This quantity is fully determined by the layer configuration.

For arithmetic cost, we count the dominant operations in scaled dot-product attention, namely the query–key dot products and the value aggregation operations used to form the attention output. At stage $t \geq 1$ (KV-cached decoding), attention is computed for the current query token against each retained KV position. For each retained position, this involves (i) a dot product between the query and key vectors and (ii) a weighted aggregation of the corresponding value vector. Both operations are performed independently for each attention head with head dimension $d_h$, and their costs aggregate across heads. Since $h_{\mathrm{q}}^{(\ell)}$ heads are used and $h_{\mathrm{q}}^{(\ell)} d_h = d$, each query–key pair accounts for $d$ MACs from the query–key dot product and an additional $d$ MACs from the value aggregation, for a total of $2d$ MACs per pair. Accordingly, we set $u_{1,\ell} := 2d$.

During prefill ($t = 0$), attention is evaluated for $\bar{n}_{\ell,0}$ query tokens against $\bar{n}_{\ell,0}$ retained key positions, producing $\bar{n}_{\ell,0}^2$ query–key pairs. For each query–key pair, scaled dot-product attention executes two dominant arithmetic steps: (i) computing the inner product between the query and key vectors, and (ii) multiplying the resulting attention weight with the corresponding value vector and accumulating the result into the output. Both steps are performed independently for each of the $h_{\mathrm{q}}^{(\ell)}$ attention heads, each operating on vectors of dimension $d_h$. Aggregating across heads, the query–key inner product accounts for $h_{\mathrm{q}}^{(\ell)} d_h = d$ MACs per pair, and the value aggregation accounts for an additional $d$ MACs per pair. Thus, each query–key pair incurs $2d$ MACs, and we set $u_{2,\ell} := 2d$.

**FFN coefficients.** Recall that we model FFN-related memory traffic and arithmetic as $\mathrm{Bytes}_{\ell,t}^{\mathrm{ffn}} = \bar{\kappa}_{\ell,t}\Big(c_{0,\ell} + c_{1,\ell}\, \bar{w}_{\ell,t}\Big)$. $\mathrm{MAC}_{\ell,t}^{\mathrm{ffn}} = v_{1,\ell}\, \bar{\kappa}_{\ell,t}\, \bar{w}_{\ell,t}$, where $\bar{\kappa}_{\ell,t}$ denotes the number of executed tokens at stage $t$ and $\bar{w}_{\ell,t}$ is the retained FFN width at layer $\ell$.

We next define the coefficient $v_{1,\ell}$, which captures the per-channel arithmetic cost (here, we count only the dominant operations in the FFN, namely the matrix–vector products in the linear projections). For a single token, the FFN maps the $d$-dimensional input activation through a set of linear projections that produce intermediate channels, followed by a linear projection back to dimension $d$. Specifically, for each active intermediate channel, the FFN performs: (i) one dot product of dimension $d$ for each input-side projection that generates that channel, and (ii) one dot product of dimension $d$ in the output projection that maps the intermediate activation back to the model dimension.

Let $\varepsilon_\ell$ denote the number of input-side projections at layer $\ell$. For a standard FFN with a single expansion projection (e.g., $\mathrm{FFN}(x) = \phi(xW_1)W_2$), we have $\varepsilon_\ell = 1$. For gated FFNs such as SwiGLU or GeGLU (e.g., $\mathrm{FFN}(x) = \phi(xW_1) \odot (xW_2)W_3$), two separate input-side projections are applied to the same $d$-dimensional input, so $\varepsilon_\ell = 2$.

Since each input-side projection involves $d$ MACs per channel, and the output projection involves an additional $d$ MACs per channel, the total arithmetic cost per active FFN channel is $(\varepsilon_\ell + 1)d$ MACs, and we have:

$$v_{1,\ell} := (\varepsilon_\ell + 1)\, d. \tag{29}$$

The coefficient $c_{1,\ell}$ captures the FFN memory traffic that scales with the number of active intermediate channels. Similarly, for each active channel, the FFN reads a $d$-dimensional weight vector from each input-side projection and a $d$-dimensional weight vector from the output projection. Thus, $(\varepsilon_\ell + 1)d$ weight elements are read per active channel, and we set:

$$c_{1,\ell} := (\varepsilon_\ell + 1)\, d\, b_{\mathrm{w}}, \tag{30}$$

where $b_{\mathrm{w}}$ denotes the number of bytes per weight element.

All FFN-related memory traffic that is independent of the effective FFN width is grouped into the constant term:

$$c_{0,\ell} := \mathrm{Bytes}_{\ell}^{\mathrm{ffn,fix}}, \tag{31}$$

where $\mathrm{Bytes}_{\ell}^{\mathrm{ffn,fix}}$ denotes the FFN-side memory traffic at layer $\ell$ that is independent of the retained FFN width. This includes, for each executed token, reading the $d$-dimensional input activation, writing the $d$-dimensional FFN output activation, accessing bias parameters, and any fixed kernel or runtime metadata required for FFN execution. The value of $\mathrm{Bytes}_{\ell}^{\mathrm{ffn,fix}}$ is fully determined by the layer configuration.

# D Experimental Supplements

## D.1 Implementation Details

**Training details.** All experiments were conducted on a server equipped with 2× Intel Xeon Platinum 8570 CPUs (112 physical cores and 224 threads total), approximately 2 TB of DDR5 system memory, with 8× NVIDIA H200 SXM5 GPUs. Computation employed PyTorch automatic mixed precision with bfloat16. Training is performed for 2 epochs with batch size ($bs$) 8. Token pruning is applied at a block granularity of $B = 8$ tokens, and FFN pruning is applied at a channel-block granularity of $G = 64$ channels. Optimization uses AdamW with learning rates $1 \times 10^{-4}$ for the token selector and $3 \times 10^{-4}$ for the FFN selector, weight decay $0.01$ applied to non-bias parameters, betas $(0.9, 0.95)$, and $\epsilon = 10^{-8}$. The global dual variable $\lambda$ is updated with learning rate $5 \times 10^{-3}$ and clipped to a maximum value of 20. Selector relaxation parameters are fixed across all experiments, with temperature $\mathcal{T} = 1$, stretch parameters $\varrho = -0.1$ and $\zeta = 1.1$. To train selectors that remain stable under different energy constraints, we employ sandwich training with $\mathcal{S} = 3$ energy budgets per mini-batch. For each mini-batch, we evaluate the model three times under distinct energy constraints: (i) a minimum budget $\mathcal{A}_{\min} = r_{\min} E_{\mathrm{dense}}(x)$ with $r_{\min} \sim \mathcal{U}(0.15, 0.35)$; (ii) a maximum budget $\mathcal{A}_{\max} = r_{\max} E_{\mathrm{dense}}(x)$ with $r_{\max} \sim \mathcal{U}(0.75, 1.0)$; and (iii) a midpoint budget $\mathcal{A}_{\mathrm{mid}} = \frac{1}{2}(\mathcal{A}_{\min} + \mathcal{A}_{\max})$. The same inputs are used for all three budgets, and the resulting losses are averaged before a single parameter update.

**Evaluation data and splits.** Unless otherwise specified, we evaluate perplexity on fixed-length sequences of length 1024 from WikiText-2-raw under teacher forcing. WikiText-2-raw is used exclusively for evaluation. No model parameters, pruning policies, or execution strategies are tuned on this dataset. For commonsense reasoning benchmarks (BoolQ, PIQA, HellaSwag, WinoGrande, ARC-Easy, ARC-Challenge, and OpenBookQA), we report accuracy on their publicly available validation splits, following standard evaluation protocols used in prior work (Anagnostidis et al., 2023; Le et al., 2025; Guo et al., 2025).

## D.2 Energy Model Calibration Details

We provide calibration details for LLaMA 3.2-3B as a representative example; the same procedure is applied independently to LLaMA 3.2-1B and LLaMA 3.1-8B on the same target device.

**Calibration sweeps.** Calibration is performed using controlled sweeps of real forward passes, in which exactly one execution variable (token retention or FFN width) is varied while all other architectural and execution parameters are kept dense and unchanged.

**Scenario A: token retention sweep.** We measure the relationship between token retention and inference energy by executing controlled forward passes of the LLaMA-3.2-3B model with different degrees of token compaction. For each dense sequence length $\mathcal{Z} \in \{128, 256, 384, 512, 768, 1024, 1536, 2048, 3072, 4096, 6144, 8192, 12288, 16384\}$, we construct packed inputs by retaining exactly $K_{\mathrm{tok}}$ token blocks of size $B$, resulting in an executed length $\mathcal{Z}_{\mathrm{keep}} = BK_{\mathrm{tok}}$. We vary $K_{\mathrm{tok}}$ over a wide range, from aggressive token pruning ($\mathcal{Z}_{\mathrm{keep}}/\mathcal{Z} = 0.05$) to the dense case, with evenly spaced intermediate keep ratios.

**Scenario B: FFN width sweep.** We fix $\mathcal{Z} = 2048$ with full token retention ($\mathcal{Z}_{\mathrm{keep}} = 2048$) and vary the FFN width in one layer at a time. Let $G = 64$ be the channel-block size and let $J = d_{\mathrm{ff}}/G$ denote the total number of FFN channel blocks per layer (for LLaMA 3.2–3B, $d_{\mathrm{ff}} = 8192$ so $J = 128$). For each layer $\ell$, we sweep an explicit set of retained block counts $K_{\mathrm{ffn},\ell} \in \{1, 2, 3, 4, 6, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 56, 64, 72, 80, 96, 112, 128\}$, implemented by passing normalized levels $K_{\mathrm{ffn},\ell}/J$ to the sweep routine. At each sweep point, only layer $\ell$ uses the reduced width $W_{\mathrm{var}} = GK_{\mathrm{ffn},\ell}$, while all other layers keep dense FFNs.

**Energy measurement protocol** We measure end-to-end inference energy consumption on device $q$ using standard power telemetry interfaces. GPU power is obtained via NVML, while CPU and DRAM energy are measured using RAPL; together, these signals capture the total device energy consumed during execution. For each sweep config-

uration, we first run warm-up forward passes and discard their measurements to stabilize kernel autotuning, memory residency, and caching effects. We then execute three measured forward passes and report the average to mitigate telemetry noise. During execution, power is sampled at 250 Hz and numerically integrated over the duration of each forward pass to obtain energy in Joules.

**Coefficient fitting via nonnegative ridge regression.** With Eq. 7, we theoretically model the end-to-end inference energy on device $q$ using analytically computed memory-traffic and arithmetic-operation counts, together with device-specific energy coefficients. We then adopt a regression approach to estimate these coefficients using on-device energy measurements collected over a short calibration sweep, in which we vary execution configurations and record the corresponding end-to-end inference energy.

For notational convenience, we index each calibration sweep configuration by $i$ and write $\widehat{E}_q(i;\Theta_q)$ to denote the model-predicted energy under that configuration. The objective is to find the optimal coefficient values that minimize the discrepancy between the model predicted energy $\widehat{E}_q(i;\Theta_q)$ and the measured energy $\mathcal{E}_i$ across all sweep points, which is formulated as:

$$\min_{\Theta_q} \frac{1}{2}\sum_i \left\|\widehat{E}_q(i;\Theta_q) - \mathcal{E}_i\right\|_2^2 + \frac{\lambda_c}{2}\|\Theta_q\|_2^2, \quad (32)$$

where $\Theta_q$ denotes the model coefficients $\{\alpha_{q,\ell}^{\mathrm{HBM}}, \alpha_{q,\ell}^{\mathrm{MAC}}, \beta_{q,\ell}^{\mathrm{HBM}}, \beta_{q,\ell}^{\mathrm{MAC}}, \gamma_q\}$, and $\lambda_c$ is a small regularization parameter used for numerical stability.

**Validation.** We validate the calibrated energy model by comparing its predictions with measured end-to-end inference energy across the full calibration sweep. Figure 1 plots predicted versus measured energy for each calibration run of the LLaMA-3.2-3B model

We summarize prediction accuracy using the coefficient of determination ($R^2$) and a weighted relative error metric. Given measured energy $y_i$ and proxy prediction $\hat{y}_i$, the coefficient of determination is defined as:

$$R^2 = 1 - \frac{\sum_i (\hat{y}_i - y_i)^2}{\sum_i (y_i - \bar{y})^2}, \quad (33)$$

where $\bar{y}$ denotes the mean of the measured energies. Because configurations with higher energy consumption contribute more to total inference energy, treating all configurations equally would underemphasize errors in high-energy regimes; therefore, we further evaluate proxy accuracy using an energy-weighted relative error, defined as

$$\mathrm{WRE} = \sum_i w_i \frac{|\hat{y}_i - y_i|}{y_i}, \qquad w_i = \frac{y_i}{\sum_j y_j}. \quad (34)$$

Across 3071 calibration samples, the proxy achieves an $R^2$ of 0.9982 and a weighted relative error of 1.99%, demonstrating accurate modeling of device energy over the operating regimes relevant to inference.

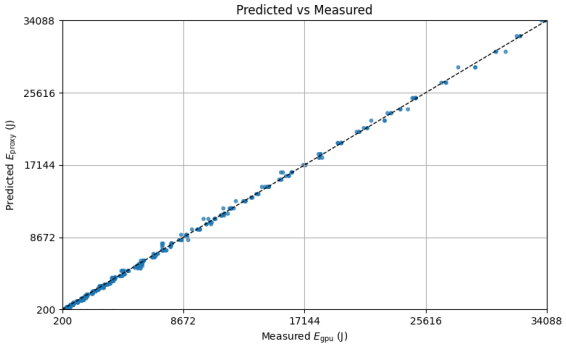


Figure 1: Overall energy calibration: predicted versus measured energy (LLaMA 3.2–3B).

We further examine the token-retention sweep (Scenario A) by plotting measured energy as a function of the token keep fraction for multiple sequence lengths, as shown in Figure 2. For each fixed $\mathcal{Z}$, energy increases monotonically with the retained length $\mathcal{Z}_{\mathrm{keep}}$, while curves corresponding to different $\mathcal{Z}$ separate cleanly, reflecting increased attention computation and KV-cache traffic. This predictable dependence on token retention confirms that the proxy correctly captures the attention-related compute and memory contributions to energy.

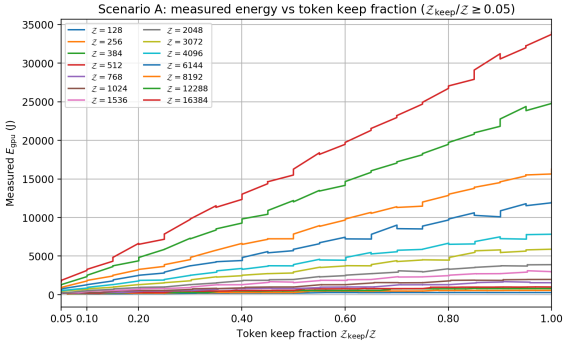


Figure 2: Scenario A (token retention): measured energy versus token keep fraction across sequence lengths (LLaMA 3.2–3B).

We next examine the FFN width sweep (Scenario B) to validate the FFN-side terms of the en-

ergy proxy. In this evaluation, the FFN width is varied in one layer at a time while all other layers execute with dense FFNs. Figure 3 plots measured energy as a function of the effective FFN width $W_{\text{var}}$, with values averaged over all layers and repeated runs that share the same $W_{\text{var}}$. As shown in this figure, energy increases monotonically with $W_{\text{var}}$ and follows an approximately linear trend. This behavior demonstrates that changes in FFN width at a single layer lead to approximately proportional changes in total energy, confirming the linear scaling behavior captured by the FFN arithmetic and memory components of the energy proxy.

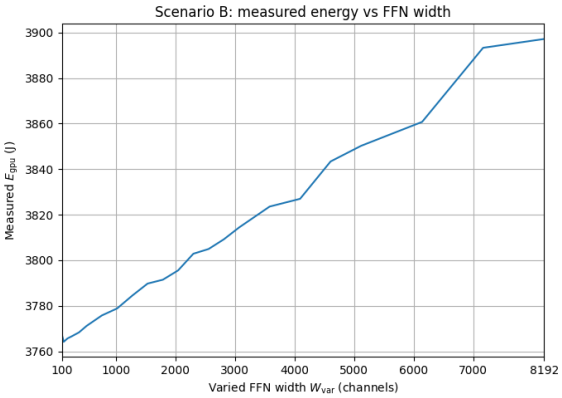


Figure 3: Scenario B (FFN width): measured energy versus effective FFN width (LLaMA 3.2–3B).

**Guard margin for conservative energy enforcement.** While the calibrated proxy closely matches measured end-to-end inference energy, small residual discrepancies remain due to finite calibration coverage and measurement noise. To ensure conservative budget enforcement under such residual errors, we introduce an explicit *guard margin* into the energy proxy. Let $\widehat{E}(x)$ denote the calibrated proxy prediction for input $x$. We define the guarded proxy as:

$$\widehat{E}^{\text{guard}}(x) \;=\; \widehat{E}(x) + \delta, \tag{35}$$

where the guard margin $\delta$ is computed from all calibration sweep configurations as

$$\delta \;=\; \Big[\max_{i \in \mathcal{C}_{\text{cal}}} \big(y_i - \widehat{E}(x_i)\big)\Big]_+. \tag{36}$$

Here $\mathcal{C}_{\text{cal}}$ denotes the set of all calibration samples collected across the token-retention and FFN-width sweeps described above. All per-sequence energy budgets during training and inference are enforced using the guarded proxy $\widehat{E}^{\text{guard}}$. As a result, execution policies that satisfy the guarded proxy constraint also satisfy the corresponding device-level energy budget within the calibrated operating regime. This conservative enforcement is also empirically verified in Section Energy Budget Adherence Distribution, which shows no device-level budget violations.

## D.3 Additional experiment results

### D.3.1 GSM8K Evaluation under Full Autoregressive Decoding

We evaluate EOP-LLM on GSM8K using LLaMA-3.2-3B-Instruct (bf16) under full autoregressive decoding. Specifically, we format inputs using the official HuggingFace chat template for LLaMA-3.2-Instruct, where each problem is presented as a user message and the model is instructed to reason step by step before producing the final answer. Decoding uses greedy search with `max_new_tokens` $=$ 512. We evaluate on the official GSM8K validation set (1,319 problems) in a no teacher forcing setting, meaning each next token is conditioned solely on the model's previously generated tokens. Dense and EOP-LLM use identical prompts and decoding configurations to ensure a controlled comparison. We enforce explicit energy budgets $\Psi \in \{0.6, 0.5\}$ on the total energy consumed during each complete generation. In Table 6, we report accuracy (ACC), the average number of generated tokens (Avg. Tokens), and the energy-budget adherence $\Delta E$ to evaluate reasoning performance, variable-length behavior, and strict budget adherence under full autoregressive decoding.

As shown in Table 6, these results demonstrate that EOP-LLM maintains stable reasoning performance under free-form autoregressive decoding with instruction-following prompts and variable-length outputs, while strictly satisfying given energy budgets.

| Method | $\Psi$ | ACC | Avg. Tokens | $\Delta E$ (%) |
|---|---|---|---|---|
| Dense | 1.0 | 76.27 | 208.39 | 0.00 |
| EOP-LLM | 0.6 | 70.13 | 132.41 | 0.22 |
| EOP-LLM | 0.5 | 65.96 | 113.73 | 0.19 |

Table 6: Average accuracy (%) over 3 trials on GSM8K using LLaMA-3.2-3B.

### D.3.2 Parameters of EOP-LLM Components

As summarized in Table 7, we report the trainable parameter overhead introduced by the auxiliary components of EOP-LLM relative to the frozen backbone. Note that, the *gating and hardening*

| Model | Token selector | FFN selector | Energy proxy | Total learned | % backbone |
|---|---|---|---|---|---|
| 1B | 0.850M | 32 | 177 | 0.850M | 0.085% |
| 3B | 1.263M | 56 | 309 | 1.264M | 0.042% |
| 8B | 1.685M | 64 | 353 | 1.686M | 0.021% |

Table 7: Trainable parameters overhead of EOP-LLM components.

*procedures* used to enforce energy budgets are algorithmic and introduce no additional trainable parameters.

**Token selector Parameters.** The token selector is implemented as a compact neural network. Across all model sizes, the token selector contains approximately $0.85$–$1.69$M parameters, scaling linearly with the backbone hidden dimension and remaining below $0.1\%$ of the backbone parameters.

**FFN selector Parameters.** The FFN selector scores FFN channel blocks using a per-layer linear map applied to a fixed, low-dimensional block feature vector. Consequently, the FFN selector introduces a small number of trainable parameters that scale linearly with the number of layers, totaling $32$, $56$, and $64$ parameters for the 1B, 3B, and 8B models, respectively.

**Energy proxy parameters.** The energy proxy is parameterized by a set of non-negative scalar coefficients estimated via ridge regression during calibration. The number of proxy parameters scales with model size, totaling $177$, $309$, and $353$ coefficients for the 1B, 3B, and 8B models, respectively.

#### D.3.3 Energy Overhead of Pruning Components

We measure the energy overhead introduced by the pruning components using a controlled two-setting comparison. First, we perform standard dense inference using the pretrained backbone alone and record the end-to-end inference energy consumption. Second, we enable all pruning components (selectors, coordination, and hardening) while fixing all execution gates to one, ensuring that all tokens and FFN channels are executed and that no physical pruning occurs, and record the corresponding end-to-end inference energy consumption. The difference between these two measurements isolates the energy overhead attributable solely to the pruning logic. We conduct this evaluation on LLaMA 3.2–3B as a representative model and measure energy after warm-up over a fixed end-to-end inference workload with sequence length 2048. The workload uses fixed subsequences extracted from the WikiText-2 (raw) test split. As shown in Table 8, the pruning components introduce an average overhead of $44.5$ J, corresponding to $1.14\% \pm 0.20\%$ of dense execution energy.

#### D.3.4 Prefill and Decode Efficiency

We evaluate prefill and decoding efficiency on LLaMA 3.2–3B under a target ratio $\Psi = 0.6$ using sequences of length 1024 and reporting end-to-end measured latency. All methods are evaluated on the WikiText-2-raw test split, using the same fixed evaluation prompts. Each measurement is repeated 10 times, and reported values are averaged over runs. Prefill latency measures the wall-clock time to process the full prompt and construct the corresponding KV cache. Decode latency is reported as seconds per decoding step with batch size $bs = 8$, where each step generates one new token per sequence using KV caching; the corresponding throughput is reported in tokens/s. As shown in Table 9, EOP-LLM reduces prefill latency to $0.557$s, achieving a $1.46\times$ speedup. At the same target ratio $\Psi = 0.6$, DCP and PP reduce prefill latency to $0.572$s ($1.42\times$) and $0.667$s ($1.22\times$), respectively. This improvement occurs because pruning decisions in EOP-LLM are applied during prefill, reducing both attention and FFN computation. In contrast, DCP prunes tokens while keeping the full FFN width, and PP incurs additional overhead from probe-based width estimation, limiting their prefill efficiency. During decoding, EOP-LLM achieves $4075.81$ tokens/s ($1.53\times$ speedup), outperforming DCP ($3970.22$ tokens/s, $1.49\times$) and PP ($4014.45$ tokens/s, $1.51\times$). Note that although PP applies a fixed pruned subnetwork during decoding without probe overhead, it retains the full token sequence, so attention-side costs continue to scale with the number of tokens, limiting decode efficiency.

#### D.3.5 Layer-Wise Execution Patterns under Energy Budgets

Using LLaMA 3.2–3B as a representative backbone, Figures 4 and 5 visualize the layer-wise exe-

| Model | Setting | Mean Energy (J) | Overhead (J) | Overhead (%) |
|---|---|---|---|---|
| LLaMA 3.2-3B | Dense (backbone only) | $3897.2 \pm 5.11$ | – | – |
| LLaMA 3.2-3B | Dense + pruning components (inactive) | $3941.7 \pm 6.09$ | $44.5 \pm 7.95$ | $1.14 \pm 0.20$ |

Table 8: Energy overhead of pruning components on LLaMA 3.2-3B after warm-up. Energy is measured on a fixed end-to-end workload with full FFN width ($W_{\text{var}} = 8192$) and sequence length 2048; values are reported as mean $\pm$ standard deviation over 5 repeated runs.

| Model | Method | $\Psi$ | $\Delta E\%$ | Prefill (s) | Decode (s/step, $bs$=8) | Tokens/s | Prefill Speedup | Decode Speedup |
|---|---|---|---|---|---|---|---|---|
| 3B | Dense | 0.0 | 0.00 | 0.812 | 0.003011 | 2656.92 | 1.00 | 1.00 |
| 3B | PP | 0.6 | 0.97 | 0.667 | 0.001993 | 4014.45 | 1.22 | 1.51 |
| 3B | DCP | 0.6 | 1.01 | 0.572 | 0.002015 | 3970.22 | 1.42 | 1.49 |
| 3B | Ours | 0.6 | 0.23 | 0.557 | 0.001963 | 4075.81 | 1.46 | 1.53 |

Table 9: Prefill and decode efficiency of LLaMA 3.2–3B under target ratio $\Psi$. Speedups are relative to the dense model and averaged over 10 runs.

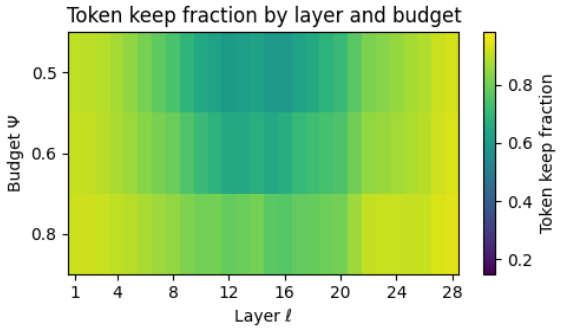


Figure 4: Layer-wise token execution under energy budgets. Heatmap shows the average fraction of executed tokens at each transformer layer $\ell$ for target energy budgets $\Psi \in \{0.5, 0.6, 0.8\}$.

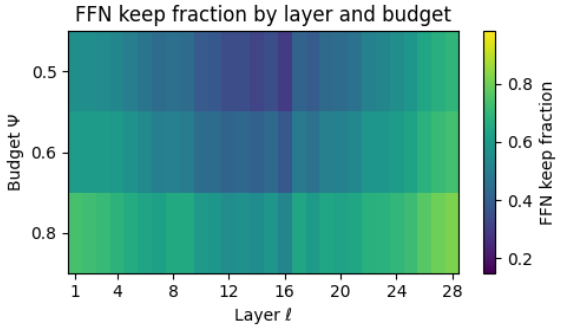


Figure 5: Layer-wise FFN execution under energy budgets. Heatmap shows the average fraction of executed FFN channels at each transformer layer $\ell$ for target energy budgets $\Psi \in \{0.5, 0.6, 0.8\}$.

cution behavior of EOP-LLM under different target energy budgets. The results are computed over the entire WikiText-2-raw test split with sequence length 1024. Each cell reports the average fraction of executed tokens (Figure 4) or executed FFN channels (Figure 5) at layer $\ell$, averaged across the full inference workload, including prefill and all autoregressive decoding steps. As the target budget tightens, execution decreases monotonically across layers, indicating that the learned policy enforces the specified energy constraints. Importantly, execution is not uniform across depth. Early and late layers are consistently preserved, whereas middle layers are pruned more aggressively. These results suggest that EOP-LLM learns depth-dependent execution policies.

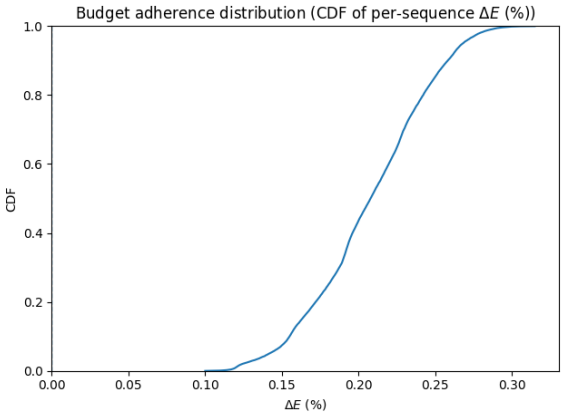


Figure 6: Budget adherence distribution (CDF). CDF of per-sequence budget slack $\Delta E$ (%) on LLaMA 3.2–3B at $\Psi = 0.6$ (WikiText-2-raw, sequence length 1024).

### D.3.6 Energy Budget Adherence Distribution

We analyze per-sequence energy budget adherence on LLaMA 3.2–3B as a representative backbone under the target budget $\Psi = 0.6$. Figures 6 and 7 present the CDF and histogram of the budget slack $\Delta E(\%)$ computed over 30,000 sequences from WikiText-2-raw. All measurements are obtained from full end-to-end inference, including both prefill and autoregressive decoding under teacher forcing with sequence length 1024.

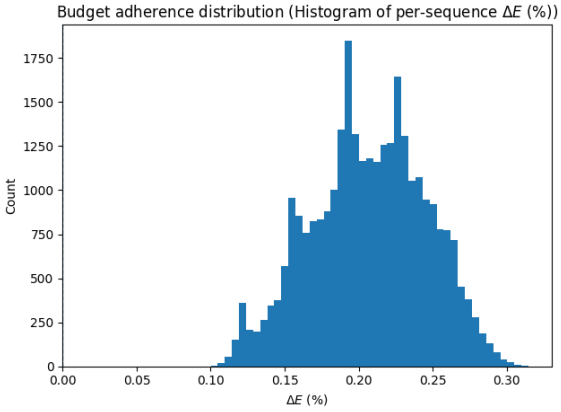


Figure 7: Budget adherence distribution (histogram). Histogram of per-sequence budget slack $\Delta E$ $(\%)$ on LLaMA 3.2–3B at $\Psi = 0.6$ (WikiText-2-raw, sequence length 1024).

Across all evaluated sequences, the target energy constraint is satisfied, with no observed violations ($\Pr[\Delta E < 0] = 0$). This observation is expected, as during both policy training and inference the execution policy is constrained using a guarded energy proxy that enforces the budget conservatively to account for residual proxy error and device-level measurement noise. Consequently, the learned policy under-utilizes the target budget slightly.