CORRELATION-AWARE STRUCTURED PRUNING FOR LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Structured pruning is a promising approach for reducing the substantial inference costs of Large Language Models (LLMs) while maintaining hardware efficiency. Existing methods often rely on independent unit scores or capture cross-unit dependencies only implicitly, which may misestimate the joint effect of pruning correlated units. To address this, we propose a Correlation-Aware Structured Pruning method. We explicitly decompose the set-level reconstruction error into individual pruning costs and pairwise interactions that capture dependencies in activations and weights, leading to a cardinality-constrained binary quadratic program. Since this binary quadratic program is NP-hard and difficult to solve exactly, we develop a greedy interaction algorithm based on dependency-aware marginal costs to optimize unit selection. Furthermore, we incorporate a gradient-based strategy to achieve adaptive layer-wise sparsity allocation across the entire model. Extensive experiments on mainstream LLMs demonstrate that incorporating correlation information yields competitive accuracy-efficiency trade-offs compared to representative structured pruning baselines.

1 INTRODUCTION

Large language models (LLMs) have advanced rapidly in recent years, demonstrating strong capabilities in instruction following, reasoning, and code generation (Brown et al., 2020; Chowdhery et al., 2023; Touvron et al., 2023a). These performance gains come with ever-increasing model sizes and inference-time resource demands. The large parameter count and activation footprint make LLM deployment challenging, imposing substantial latency, memory, and energy costs—especially in resource-constrained settings.

To reduce the cost of LLM inference, a variety of model compression techniques (Zhu et al., 2024) have been explored, including pruning (Ma et al., 2023; Sun et al., 2024; Frantar & Alistarh, 2023), quantization (Frantar et al., 2023; Xiao et al., 2023), and knowledge distillation (Xu et al., 2024). Among them, pruning directly reduces model complexity by removing redundant weights or architectural units. In particular, structured pruning improves inference efficiency by removing high-level architectural units such as attention heads or channels while preserving dense computation patterns, making it more hardware-friendly than unstructured pruning (Ma et al., 2023).

Recent advancements in structured pruning for LLMs have focused on the post-training regime, with several representative methods emerging. LLM-Pruner (Ma et al., 2023) ensures structural consistency by identifying coupled units via dependency graph analysis and pruning them based on gradient sensitivity. OSSCAR (Meng et al., 2024) jointly optimizes structured pruning decisions and retained-weight reconstruction through a layer-wise combinatorial formulation. LoRAP (Li et al., 2024) introduces a hybrid paradigm: attention weights suit low-rank decomposition, while FFN layers favor structured channel pruning. Furthermore, recent studies have proposed novel metrics to capture the intrinsic redundancy in model weights—SlimGPT (Ling et al., 2024) extends Optimal Brain Surgeon (OBS) to structured pruning, and FLAP (An et al., 2024) uses a fluctuation-based metric for feature stability to guide channel pruning.

Many existing structured pruning methods rank prunable units (e.g., attention heads or channels) independently using scalar importance scores (Ma et al., 2023; Li et al., 2024; An et al., 2024), implicitly assuming additive pruning costs. However, non-orthogonal weights and correlated activations introduce cross-unit interactions, potentially reinforcing or canceling each other. Independent ranking

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

can therefore yield suboptimal pruning masks. While some existing methods (Ling et al., 2024; Meng et al., 2024) account for cross-unit dependencies through second-order or combinatorial reconstruction objectives, these dependencies remain implicit rather than being explicitly characterized at the pruning-unit level. Consequently, pairwise error amplification or cancellation is not directly exposed during unit selection. This motivates an explicit formulation that directly derives pairwise interactions from the local pruning error.

In this paper, we propose Correlation-Aware Structured Pruning (CASP) for LLMs. We start from a reconstruction-based local pruning objective. By expanding the reconstruction error for pruning a set of units, we find it naturally decomposes into two parts: a diagonal part for conventional per-unit importance and an off-diagonal part capturing cross-unit dependencies from weight alignment and activation correlations. Building on this objective, we formulate a binary quadratic program and develop a greedy interaction algorithm based on dependency-aware marginal costs for efficient mask generation. To enhance end-to-end performance, we further adopt a gradient-based layer-wise allocation strategy that adaptively adjusts compression rates according to the sensitivity of each layer. Experiments on mainstream LLMs demonstrate that incorporating cross-unit dependency information yields more robust pruning decisions and better accuracy–efficiency trade-offs than representative structured pruning baselines at the same sparsity level.

Our contributions are summarized as follows:

- **Correlation-aware modeling.** We derive the exact local set-pruning reconstruction error under fixed calibration activations and formulate structured pruning as a cardinality-constrained binary quadratic program that explicitly models channel dependencies.
- **Greedy Interaction Algorithm.** We design a greedy algorithm for the resulting quadratic program using interaction-aware marginal-cost updates, and adapt it to attention head pruning via block-wise aggregation.
- **Gradient-Based Sparsity Allocation.** We propose a gradient-based layer-wise allocation strategy that allocates sparsity according to layer sensitivity under a fixed global sparsity level.

2 PRELIMINARIES

2.1 TRANSFORMER MECHANISMS

We use LLaMA-style decoder-only Transformers as an example and define the notation for a standard Transformer block, which consists of a multi-head self-attention (MHA) sublayer and a gated feed-forward network (FFN) sublayer. Let $X \in \mathbb{R}^{L\times d}$ denote the input activation, where L and d are the sequence and hidden dimensions, respectively.

Given h attention heads and per-head dimension d_h with $d = hd_h$, the i-th attention head applies linear projections to obtain query, key, and value representations:

$$Q^{(i)} = XW_q^{(i)}, \quad K^{(i)} = XW_k^{(i)}, \quad V^{(i)} = XW_v^{(i)},$$

where $W_q^{(i)}, W_k^{(i)}, W_v^{(i)} \in \mathbb{R}^{d\times d_h}$ are learnable projection matrices. The head output is computed by scaled dot-product attention:

$$\mathrm{head}_i = \mathrm{Softmax}\left(\frac{Q^{(i)}{K^{(i)}}^\top}{\sqrt{d_h}}\right) V^{(i)}.$$

The multi-head attention concatenates all head outputs along the feature dimension and projects them back to $\mathbb{R}^{L\times d}$:

$$\mathrm{MHA}(X) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_h)\, W_o,$$

where $W_o \in \mathbb{R}^{d\times d}$ is the output projection.

The FFN expands d to an intermediate width d_m and projects back to d through an element-wise gated activation:

$$\mathrm{FFN}(X) = \big((XW_{\text{up}}) \odot \sigma(XW_{\text{gate}})\big)\, W_{\text{down}},$$

where $W_{\text{up}}, W_{\text{gate}} \in \mathbb{R}^{d \times d_m}$ project X into the intermediate space and $W_{\text{down}} \in \mathbb{R}^{d_m \times d}$ maps it back to the hidden dimension. Here $\odot$ denotes element-wise multiplication, and $\sigma(\cdot)$ is the SiLU activation.

2.2 Layer-Wise Pruning

Structured pruning removes entire architectural units to obtain hardware-friendly acceleration. For LLMs, it is commonly applied along four dimensions: attention heads, FFN intermediate neurons (channels), embedding width, and model depth. In this work, we focus on pruning (i) attention heads in the MHA sublayer and (ii) intermediate channels in the gated FFN, allowing non-uniform sparsity across layers.

A unified view is to cast structured pruning as selecting a subset of rows in a linear map. Let $Y \in \mathbb{R}^{L \times d_m}$ denote the input to a projection matrix $W \in \mathbb{R}^{d_m \times d}$, so the dense output is $Z = YW$. We introduce a binary mask $M \in \{0,1\}^{d_m \times d}$ (or row-wise mask) to zero out selected structures in W, and define the reconstruction objective

$$\min_{M} \left\| YW - Y\big(M \odot W\big) \right\|_F^2.$$

This objective instantiates to the two targets in this paper: (1) **Attention-head pruning.** Writing the MHA output as $O = HW_O$ (with H the concatenation of head outputs), pruning a head corresponds to masking the associated block of the output projection W_O, i.e., $Y = H$ and $W = W_O$. (2) **FFN channel pruning.** For the gated FFN, the down-projection takes $Z = YW_{\text{down}}$ with $Y = (XW_{\text{up}}) \odot \sigma(XW_{\text{gate}})$; pruning an intermediate channel corresponds to masking the corresponding row(s) in W_{down}, i.e., Y is the intermediate activation and $W = W_{\text{down}}$. Once heads/channels are selected for pruning, the corresponding parameters in the preceding projections are pruned accordingly (e.g., $W_q^{(i)}, W_k^{(i)}, W_v^{(i)}$ for attention heads, and $W_{\text{up}}, W_{\text{gate}}$ for FFN channels).

Existing methods (Sun et al., 2024; Ma et al., 2023; Li et al., 2024) typically assign an importance score to each prunable unit independently (e.g., based on reconstruction error energy combining weight magnitude and activation energy) and retain units with the highest scores.

2.3 Notations

Building on the linear map $Z = YW$ introduced above, we decompose the output as $Z = \sum_{j=1}^{d_m} y_j w_j^\top$, where y_j denotes the j-th column of Y and $w_j^\top$ denotes the j-th row of W. Let $S \subseteq \{1, \ldots, d_m\}$ be the set of k pruned indices. The pruned output is then given by $\hat{Z} = \sum_{j \notin S} y_j w_j^\top$. The reconstruction error is quantified via the expected squared Frobenius norm (Ling et al., 2024):

$$\mathcal{L}(S) = \mathbb{E}\Big[\|Z - \hat{Z}\|_F^2\Big] = \mathbb{E}\left[\left\|\sum_{i \in S} y_i w_i^\top\right\|_F^2\right]. \tag{1}$$

3 Related Work

Pruning Granularity. Model pruning approaches can be categorized by their granularity: unstructured pruning (Frantar & Alistarh, 2023; Sun et al., 2024) removes individual weights, while structured pruning removes coherent architectural units (e.g., layers, blocks, attention heads, FFN channels) for hardware-friendly acceleration. Structured pruning spans from coarse-grained layer removal (Men et al., 2025; Sieberling et al., 2025) to finer-grained channel and head-wise pruning (Ma et al., 2023; Guo et al., 2025). Recent works focus on channel and head-wise pruning, which balances latency reduction and model capacity preservation.

Pruning Criteria. Existing pruning criteria include magnitude- and activation-based heuristics (Sun et al., 2024; An et al., 2024), gradient-based sensitivity methods (Ma et al., 2023; Zhang et al., 2024), and reconstruction-based objectives (Frantar & Alistarh, 2023; Ling et al., 2024). Many ranking-based methods score units independently, potentially overlooking joint pruning effects. Recent methods such as SlimGPT and OSSCAR (Ling et al., 2024; Meng et al., 2024) move beyond independent

scoring by exploiting second-order information or combinatorial reconstruction objectives. OSSCAR is most closely related to our work, formulating structured pruning as a layer-wise combinatorial reconstruction problem with local search. The key distinction is that OSSCAR captures cross-unit dependencies only implicitly through its reconstruction objective, whereas CASP explicitly derives a pairwise interaction matrix from the joint pruning error, enabling dependency-aware marginal-cost updates and interpretation of error amplification or cancellation.

Layer-wise Allocation. To address varying sensitivities across Transformer layers, recent methods employ non-uniform allocation strategies instead of uniform pruning. SlimLLM (Guo et al., 2025) adjusts per-layer pruning rates based on representation change, while LoRAP (Li et al., 2024) advocates for differentiated compression across sublayers. Additionally, SlimGPT (Ling et al., 2024) investigates incremental, non-uniform pruning ratios to mitigate performance degradation in high compression regimes. These works show that layer-wise allocation is critical, especially under high compression.

4 METHODOLOGY

To address the limitations of independent unit scoring, we propose the Correlation-Aware Structured Pruning method: key components include explicitly modeling cross-unit dependencies and designing a greedy interaction algorithm to handle the resulting optimization problem. Complementing these components, we incorporate a gradient-based layer-wise sparsity allocation strategy as a preprocessing step, which adaptively determines pruning ratios across layers. Together, these three components enable performance-preserving pruning of LLMs, with the full execution pipeline detailed in Figure 1.

4.1 CORRELATION-AWARE MODELING

To explicitly account for cross-unit dependencies in structured pruning, we formulate the pruning task as a constrained optimization problem:

$$\begin{aligned} \min_{z\in\{0,1\}^{d_m}} \quad & z^\top Q z \\ \text{s.t.} \quad & \mathbf{1}^\top z = k, \end{aligned} \tag{2}$$

where k denotes the number of units to be pruned.

This quadratic formulation is obtained by expanding the expected reconstruction error $\mathcal{L}(S)$ in Eq. (1). By expanding the squared Frobenius norm, we decompose the error into individual and interactive components:

$$\begin{aligned} \mathcal{L}(S) &= \sum_{i\in S}\sum_{j\in S}\mathbb{E}\Big[\big\langle y_i w_i^\top,\ y_j w_j^\top\big\rangle_F\Big] \\ &= \underbrace{\sum_{i\in S}\|w_i\|_2^2\cdot\mathbb{E}\big[\|y_i\|_2^2\big]}_{\text{diagonal / individual terms}} \quad + \underbrace{\sum_{\substack{i,j\in S\\ i\neq j}}(w_i^\top w_j)\cdot\mathbb{E}\big[y_i^\top y_j\big]}_{\text{off-diagonal / interaction terms}}. \end{aligned} \tag{3}$$

Let $z\in\{0,1\}^{d_m}$ denote the binary indicator for pruned units with budget k, corresponding to the index set $S=\{i : z_i = 1\}$. We then define the interaction matrix $Q\in\mathbb{R}^{d_m\times d_m}$ via

$$Q_{ij} \triangleq (w_i^\top w_j)\cdot\mathbb{E}\big[y_i^\top y_j\big], \tag{4}$$

which leads to the optimization problem formulated in Eq. (2).

Computationally, Q can be efficiently obtained via element-wise operations as $Q = G\odot C$, where $G = WW^\top$ captures the weight geometry and $C = \mathbb{E}[Y^\top Y]$ captures activation correlations estimated from a small calibration set.

This interaction matrix Q offers a precise physical interpretation of how pruning decisions impact model performance:

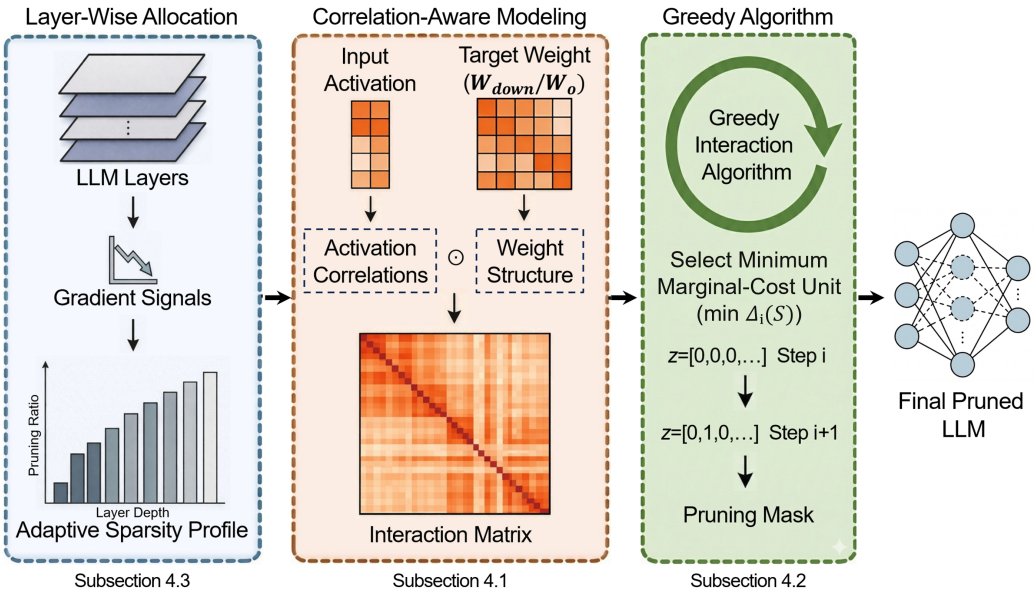

Figure 1: **The Correlation-Aware Structured Pruning (CASP) workflow.** Linear pipeline (from left to right): (1) Layer-Wise Allocation generates a sparsity profile; (2) Correlation-Aware Modeling builds the interaction matrix Q; (3) Greedy Algorithm selects optimal units to yield the pruned LLM.

- **Diagonal Terms (Q_{ii}):** These terms quantify the error induced by pruning unit i in isolation, corresponding to standard independent importance scores found in existing literature.
- **Off-Diagonal Terms ($Q_{ij}, i \neq j$):** These terms measure the *interaction* between units. A positive value ($Q_{ij} > 0$) indicates error amplification (pruning i and j together induces a greater reconstruction error than the sum of the errors induced by pruning each unit individually). Conversely, a negative value ($Q_{ij} < 0$) indicates error cancellation, suggesting that the errors from removing these units partially offset each other, thereby reducing the overall pruning cost.

Most existing structured pruning methods assess unit importance independently, which implicitly amounts to neglecting the off-diagonal terms of Q. This treatment relies on the assumption that different units are effectively decoupled, requiring the corresponding weight vectors to be orthogonal and hidden-state dimensions to be uncorrelated. However, these conditions are rarely satisfied in modern LLMs, where dense-layer weights are non-orthogonal and hidden states exhibit pronounced inter-dimensional correlations (Ethayarajh, 2019). Consequently, explicitly considering the off-diagonal terms of Q becomes essential. We further quantify in Appendix B.2 the cumulative contribution omitted by treating pruning units independently, showing that it grows rapidly with sparsity and can exceed the diagonal contribution under aggressive pruning.

The formulation in Eq. (2) corresponds to a 0–1 quadratic program, which is NP-hard in general (Garey & Johnson, 2002). Given the massive dimensionality of LLM layers (where d_m can exceed 10,000), obtaining an exact global solution is computationally intractable. This necessitates the use of an efficient approximation strategy, which we detail in the following subsection.

4.2 Greedy Interaction Algorithm

To solve the combinatorial optimization problem formulated in Eq. 2, we start with a key observation: the objective function exhibits an incremental structure, where the marginal cost of pruning a unit can be exactly derived from the current pruned set. This observation naturally leads to a greedy strategy: we construct the pruning mask sequentially, selecting at each step the unit that minimizes the incremental reconstruction error. Unlike conventional greedy pruning, which relies on fixed

Algorithm 1 Greedy Interaction Search

Input: interaction matrix $Q \in \mathbb{R}^{d_m \times d_m}$, prune count k
$S \leftarrow \emptyset$; $u \leftarrow \mathbf{0} \in \mathbb{R}^{d_m}$
for $t = 1$ **to** k **do**
  Compute marginal costs $\Delta_i \leftarrow Q_{ii} + 2u_i$ for all $i \notin S$
  Select $i^\star \leftarrow \arg\min_{i \notin S} \Delta_i$
  $S \leftarrow S \cup \{i^\star\}$
  $u \leftarrow u + Q_{:,i^\star}$
end for
Output: pruned index set S (indicator z)

importance scores, our strategy updates marginal costs based on previously selected units, enabling the algorithm to explicitly capture cross-unit dependencies.

Motivated by this observation, we propose an efficient Greedy Interaction Algorithm. Starting from an empty pruned set $S = \emptyset$, we iteratively add the unit that minimizes the increase in reconstruction error. For a candidate unit $i \notin S$, the marginal increase in the objective upon adding i to S is:

$$\Delta_i(S) = \mathcal{L}(S \cup \{i\}) - \mathcal{L}(S) = Q_{ii} + 2\sum_{j \in S} Q_{ij}. \tag{5}$$

At each step t:

- Select the optimal unit $i^\star = \arg\min_{i \notin S} \Delta_i(S)$.
- Update $S \leftarrow S \cup \{i^\star\}$.

This process repeats until the budget $|S| = k$ is satisfied. The complete procedure is formalized in Algorithm 1.

Remark 1. *We reduce the per-layer complexity from $O(k^2\, d_m)$ (naive computation) to $O(k\, d_m)$ by maintaining a running vector $u \in \mathbb{R}^{d_m}$ with $u_i = \sum_{j \in S} Q_{ij}$. This avoids recomputing the summation in Eq. 5 from scratch, allowing $\Delta_i(S) = Q_{ii} + 2u_i$ to be computed in $O(1)$ per candidate. After selecting $i^\star$, we update $u \leftarrow u + Q_{:,i^\star}$ in $O(d_m)$ time, thereby reducing the computational cost from quadratic to linear with respect to the cardinality constraint k.*

Remark 2. *The proposed interaction modeling introduces quadratic scaling with the module dimension d_m, as constructing $Q = (WW^\top) \odot \mathbb{E}[Y^\top Y]$ requires forming $d_m \times d_m$ interaction matrices. This cost is incurred only once during offline pruning and does not affect inference. The construction mainly involves GPU-friendly matrix multiplications and element-wise operations. As shown in Appendix B.4, the entire pruning procedure for LLaMA2-7B completes within a few minutes.*

Remark 3. *Our method naturally extends to coarse-grained pruning, specifically for attention heads (where the pruning unit is a block of indices rather than a single channel). Let $\{\mathcal{B}_h\}_{h=1}^{H}$ be a partition of $\{1, \ldots, d_m\}$ where $\mathcal{B}_h$ contains the indices belonging to block h. By aggregating the interaction matrix Q over these blocks, we obtain a reduced matrix $\widetilde{Q} \in \mathbb{R}^{H \times H}$:*

$$\widetilde{Q}_{ab} \triangleq \sum_{i \in \mathcal{B}_a} \sum_{j \in \mathcal{B}_b} Q_{ij},$$

and the objective simplifies to $s^\top \widetilde{Q}\, s$. The same greedy interaction search can be applied to select k heads, with marginal costs computed analogously.

Remark 4. *Theoretically, problem* (2) *can be approached by relaxing the binary constraint to $z \in [0, 1]^{d_m}$. Since Q is positive semi-definite, this relaxation constitutes a convex Quadratic Program (QP). However, direct optimization of this QP is computationally expensive for LLM layers due to their massive dimensionality, so we focus on the greedy algorithm in this work. Nevertheless, this relaxation still provides a rigorous theoretical lower bound for the discrete problem, allowing us to assess the optimality gap of our greedy solution, and can also serve as a baseline in experiments to validate the effectiveness of our method. We provide the detailed formulation and experimental comparisons in Appendix A.1 for interested readers.*

4.3 Gradient-Based Compression Allocation

This subsection introduces a gradient-based layer-wise compression allocation strategy as a pre-processing step for the pruning pipeline. This approach is designed to bridge the gap between minimizing local errors and optimizing end-to-end behavior. The process operates by first defining a layer sensitivity metric using gradient signals, and then normalizing these sensitivities to calculate layer-wise sparsity.

To bridge the gap between local pruning decisions and global model performance, we must assess how perturbations in each layer affect the final objective. We use the standard next-token cross-entropy loss as the end-to-end objective $\mathcal{J}$. Motivated by the first-order Taylor expansion, the change in $\mathcal{J}$ induced by a perturbation ΔZ_ℓ to the ℓ-th layer's output can be approximated as:

$$\mathcal{J}(Z_\ell + \Delta Z_\ell) - \mathcal{J}(Z_\ell) \approx \left\langle \frac{\partial \mathcal{J}}{\partial Z_\ell}, \Delta Z_\ell \right\rangle.$$

Since the goal of local pruning is to minimize $\|\Delta Z_\ell\|_F^2$, we evaluate the end-to-end sensitivity of each layer using the expected squared Frobenius norm of the gradient:

$$S_\ell \triangleq \mathbb{E}\left[\left\| \frac{\partial \mathcal{J}}{\partial Z_\ell} \right\|_F^2 \right].$$

A larger S_ℓ indicates higher sensitivity, implying that a lower pruning ratio should be allocated to this layer.

We observe a general rule that the magnitude of layer-wise gradients decreases as the layer depth increases (experimental results are presented in Appendix B.3). This implies that deeper layers can tolerate higher compression, motivating us to adopt an increasing pruning ratio across layers, which is similar in spirit to the layer-wise allocation strategy in SlimGPT (Ling et al., 2024).

Furthermore, a normalization scheme is developed to transform the sensitivity signals into concrete layer-wise sparsity ratios under a fixed global sparsity level, which will be presented in detail in Appendix A.4.

5 Experiments

5.1 Experimental Settings

Models and Evaluation. We evaluate our method on representative open-source large language models, including the LLaMA2 family (7B and 13B) (Touvron et al., 2023b), Vicuna-7B (Chiang et al., 2023), and the recent GQA-based LLaMA3-8B model (Grattafiori et al., 2024). To comprehensively assess the performance of the compressed models, we employ two standard metrics. First, we report the zero-shot perplexity (PPL) on the test split of WikiText2 (Merity et al., 2017) and the validation split of PTB (Marcus et al., 1993) to evaluate language modeling performance. Second, following previous structured pruning works (Ma et al., 2023; Li et al., 2024), we evaluate the zero-shot accuracy on seven commonsense reasoning datasets, including BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-easy (Clark et al., 2018), ARC-challenge (Clark et al., 2018), and OpenBookQA (Mihaylov et al., 2018).

Calibration data and implementation details. To estimate the activation statistics, we randomly sample 128 sequences from the WikiText2 (Merity et al., 2017) training set, each with a context length of 2048 tokens. The same calibration sequences are used both to construct the interaction matrices and to compute the layer-output gradients for sensitivity-based sparsity allocation. Similarly, we employ a sequence length of 2048 tokens when measuring perplexity on the test datasets. We evaluate our method across four distinct sparsity levels: 10%, 20%, 30%, and 50%. Additional allocation hyperparameters are reported in Appendix B.1.

Baselines. We compare our proposed method with the following representative structured pruning baselines:

Table 1: Zero-shot performance of the compressed LLaMA2-7B. The average score is computed across seven datasets. Bold values denote the best result under the same pruning ratio.

Ratio	Method	WikiText2 ↓	PTB ↓	BoolQ ↑	PIQA ↑	HellaSwag ↑	WinoGrande ↑	ARC-e ↑	ARC-c ↑	OBQA ↑	Average ↑
0%	Ground Truth	5.4690	19.5973	0.7936	0.7873	0.7615	0.6946	0.7382	0.4505	0.4420	0.6668
10%	LLM-Pruner	7.1563	29.9688	0.6352	0.7791	0.7294	0.6725	0.7016	0.4326	0.4160	0.6238
	OSSCAR	6.2816	23.2605	0.7098	0.7813	0.7402	0.6835	0.7302	0.4454	0.4280	0.6455
	FLAP	6.0234	23.0781	0.7187	0.7688	0.7213	0.6590	0.6831	0.4113	0.4200	0.6260
	LoRAP	5.9976	24.6672	0.7557	0.7802	0.7439	0.6819	0.7155	0.4360	0.4300	0.6490
	SlimGPT	6.1304	24.3635	0.7196	0.7758	0.7475	**0.6969**	0.7058	0.4309	0.4200	0.6424
	SlimLLM	6.4375	23.0781	0.7373	**0.7878**	0.7410	0.6756	0.7277	0.4445	0.4280	0.6488
	CASP	**5.8915**	**21.4567**	**0.7627**	0.7856	**0.7533**	0.6732	**0.7348**	**0.4590**	**0.4360**	**0.6578**
20%	LLM-Pruner	10.5469	42.7813	0.6517	0.7628	0.6882	0.6251	0.6406	0.3933	0.4220	0.5977
	OSSCAR	7.0327	26.6744	0.6899	0.7612	0.7127	**0.6953**	**0.6995**	**0.4352**	0.4220	0.6308
	FLAP	7.1211	28.0000	0.5872	0.7470	0.6538	0.6393	0.6082	0.3601	0.3980	0.5705
	LoRAP	6.9239	34.8910	0.7049	**0.7633**	0.7178	0.6701	0.6721	0.3968	**0.4280**	0.6219
	SlimGPT	7.2878	29.6757	0.6963	0.7530	0.7207	0.6882	0.6991	**0.4352**	0.4180	0.6301
	SlimLLM	8.2109	31.2969	**0.7141**	0.7584	0.7023	0.6511	0.6864	0.4147	0.4120	0.6199
	CASP	**6.7034**	**24.5221**	0.7135	0.7612	**0.7276**	0.6848	0.6888	0.4313	0.4220	**0.6327**
30%	LLM-Pruner	15.3125	68.1875	0.4676	0.7454	0.6066	0.5667	0.5509	0.3200	0.3980	0.5222
	OSSCAR	8.1444	30.6898	0.5740	0.7296	0.6545	0.6283	0.6136	0.3720	0.3900	0.5660
	FLAP	9.0156	37.4375	0.5676	0.7040	0.5792	0.6259	0.5585	0.3336	0.3880	0.5367
	LoRAP	7.8088	65.3891	**0.7061**	0.7383	0.6745	0.6440	0.6246	0.3703	**0.4200**	0.5968
	SlimGPT	8.0929	36.8019	0.6520	0.7285	0.6733	**0.6661**	0.6582	0.3951	0.4040	0.5967
	SlimLLM	10.8359	39.4688	0.6823	0.7242	0.6539	0.6298	0.6115	0.3882	0.3980	0.5840
	CASP	**7.6920**	**26.3622**	0.6942	**0.7514**	**0.6893**	0.6259	**0.6608**	**0.4232**	**0.4200**	**0.6092**
50%	LLM-Pruner	97.9563	377.5000	0.5428	0.6017	0.3427	0.5051	0.3325	0.2585	0.2940	0.4110
	OSSCAR	20.4315	60.9696	0.4248	0.5147	0.2657	0.4854	0.2769	0.2398	0.2760	0.3548
	FLAP	24.9531	143.2500	0.3847	0.6192	0.4140	0.5391	0.3666	0.2705	0.3240	0.4169
	LoRAP	23.9369	282.5295	0.6104	0.6061	0.4265	0.5549	0.3746	0.2696	0.3100	0.4503
	SlimGPT	18.3618	62.3579	0.4275	0.6322	**0.4982**	**0.6030**	**0.5210**	**0.3191**	**0.3620**	0.4804
	SlimLLM	50.7812	142.1250	**0.6303**	0.6159	0.4295	0.5462	0.4331	0.3072	0.2940	0.4652
	CASP	**15.2595**	**56.6118**	0.5924	**0.6676**	0.4949	0.5627	0.4764	0.3114	0.3460	**0.4931**

- **LLM-Pruner** (Ma et al., 2023) is the pioneering structured pruning method for LLMs. It utilizes first-order Taylor expansion to estimate the importance of coupled structures and removes the least important groups based on dependency detection.
- **OSSCAR** (Meng et al., 2024) formulates structured pruning as a layer-wise combinatorial reconstruction problem and employs efficient local search with low-rank updates for structure selection.
- **LoRAP** (Li et al., 2024) proposes a differentiated compression strategy. It applies low-rank approximation to the Multi-Head Attention (MHA) layers and gradient-free structured pruning to the Feed-Forward Networks (FFN) based on activation-weighted importance.
- **FLAP** (An et al., 2024) introduces a fluctuation-based pruning metric. It evaluates feature importance by measuring the insensitivity of the model to the removal of specific channels, focusing on preserving features with high variance.
- **SlimGPT** (Ling et al., 2024) extends the Optimal Brain Surgeon framework to structured pruning and introduces Batched Greedy Pruning for efficient approximate local optimization.
- **SlimLLM** (Guo et al., 2025) uses Pearson similarity for attention-head pruning and a feature-space criterion considering both weight magnitude and direction for FFN channel pruning, followed by lightweight linear-regression-based recovery.

Remark 5. *To rigorously assess the intrinsic quality of the pruning masks identified by CASP, we primarily focus our evaluation on the post-training regime. For completeness, we provide supplementary results demonstrating the performance recovery after fine-tuning in Appendix B.6, confirming that our method also serves as a robust initialization for further optimization.*

5.2 Zero-Shot Performance

Table 1 summarizes the zero-shot performance results for LLaMA2-7B. Additional results for LLaMA2-13B, Vicuna-7B, and the GQA-based LLaMA3-8B are provided in Appendix B.8. We compare CASP against representative baselines across varying pruning ratios. CASP achieves the best perplexity and average accuracy at all evaluated pruning ratios on LLaMA2-7B.

In low compression regimes (10% and 20%), CASP preserves strong performance while consistently reducing perplexity relative to the baselines. Its advantage becomes more evident at 50% pruning, where CASP obtains a WikiText2 PPL of 15.26 and an average zero-shot accuracy of 0.4931, compared with 18.36 and 0.4804 for SlimGPT, the strongest baseline by average accuracy. Overall, these results demonstrate that CASP offers a consistently stronger accuracy–perplexity trade-off and remains particularly robust under aggressive structured pruning.

5.3 Ablation Study

To systematically assess the contribution of each component in the proposed CASP framework, we conduct ablation studies on LLaMA2-7B at 30% and 50% pruning ratios, separately examining interaction-aware unit selection and layer-wise sparsity allocation.

Interaction-Aware Selection. Table 2 compares CASP with **Diag**, which retains only the diagonal entries of Q and therefore scores pruning units independently. CASP improves all three metrics at both pruning ratios, confirming the value of modeling local interactions during unit selection.

Table 2: Ablation of interaction-aware unit selection on LLaMA2-7B. Bold values denote the best results at each pruning ratio.

Ratio	Method	WikiText2 ↓	PTB ↓	Average ↑
30%	Diag	8.5853	33.4522	0.5739
	CASP	**7.6920**	**26.3622**	**0.6092**
50%	Diag	20.5669	76.9331	0.4433
	CASP	**15.2595**	**56.6118**	**0.4931**

Layer-Wise Allocation. Table 3 compares four ways of distributing the global pruning budget across layers. Uniform assigns the same sparsity to every layer. Log-Increasing follows the allocation rule used in SlimGPT (Ling et al., 2024), with sparsity increasing logarithmically along model depth. Cosine Similarity follows SlimLLM (Guo et al., 2025), using the similarity between each layer's input and output representations to determine its sparsity. Gradient Sensitivity is our strategy, which assigns higher sparsity to less sensitive layers. It achieves the lowest perplexity and highest average accuracy at both pruning ratios, supporting the effectiveness of sensitivity-aware allocation.

Table 3: Ablation of layer-wise sparsity schedules on LLaMA2-7B. Only the allocation rule is varied while the remaining pruning procedure is fixed.

Ratio	Method	WikiText2 ↓	PTB ↓	Average ↑
30%	Uniform	8.3228	40.8278	0.5713
	Log-Increasing	7.9370	29.7225	0.5894
	Cosine Similarity	8.9313	31.5706	0.5864
	Gradient Sensitivity	**7.6920**	**26.3622**	**0.6092**
50%	Uniform	30.9006	121.4272	0.3605
	Log-Increasing	16.1327	58.6871	0.4248
	Cosine Similarity	42.4071	147.9380	0.4517
	Gradient Sensitivity	**15.2595**	**56.6118**	**0.4931**

6 Conclusion

In this paper, we propose Correlation-Aware Structured Pruning (CASP), a post-training approach designed to address the limitations of the independence assumption commonly adopted in structured pruning. By formulating the pruning objective as a cardinality-constrained binary quadratic program, we explicitly model the cross-unit dependencies inherent in LLMs. We further develop a greedy interaction algorithm for efficient optimization, together with a gradient-based layer-wise strategy for sensitivity-aware sparsity assignment. Extensive experiments on mainstream LLMs demonstrate that CASP achieves robust performance preservation and favorable accuracy-efficiency trade-offs, particularly under high sparsity ratios.

AI USE STATEMENT

Generative AI tools were used to assist with polishing language and formatting author-provided experimental results. They were not used to develop the scientific ideas, theoretical formulation, methodology, experimental design, or experimental data. The authors checked all AI-assisted text and tables against the underlying derivations and source results and take responsibility for the final content of this work.

ETHICS STATEMENT

This work uses publicly available pretrained models and benchmark datasets and does not involve human subjects or personal or sensitive data. Structured pruning may reduce the computational and energy costs of deploying LLMs, but a compressed model may retain or alter undesirable behaviors inherited from its source model. Such models should therefore undergo application-specific safety and robustness evaluation before deployment.

REPRODUCIBILITY STATEMENT

The main text provides the pruning objective, optimization algorithm, layer-wise allocation strategy, calibration setting, baselines, and evaluation protocol. The appendix provides additional derivations, complexity analysis, implementation details, ablations, and model-specific experiments. All pretrained models and evaluation datasets used in this work are publicly available, and the reported results include the individual scores for all seven zero-shot tasks to facilitate verification.

REFERENCES

Yongqi An, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. Fluctuation-based adaptive structured pruning for large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 10865–10873, 2024.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Wei-Lin Chiang, Zhuohan Li, Ziqing Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6, 2023.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, 2019.

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

Kawin Ethayarajh. How contextual are contextualized word representations. *Comparing the geometry of BERT, ELMo, and GPT-2 Embeddings*, 2, 2019.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, pp. 10323–10337. PMLR, 2023.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan-Adrian Alistarh. Optq: Accurate post-training quantization for generative pre-trained transformers. In *11th International Conference on Learning Representations*, 2023.

Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Jialong Guo, Xinghao Chen, Yehui Tang, and Yunhe Wang. Slimllm: Accurate structured pruning for large language models. In *International Conference on Machine Learning*, pp. 20766–20776. PMLR, 2025.

Mikhail K Kozlov, Sergei Pavlovich Tarasov, and Leonid Genrikhovich Khachiyan. Polynomial solvability of convex quadratic programming. In *Doklady akademii nauk*, volume 248, pp. 1049–1051. Russian Academy of Sciences, 1979.

Guangyan Li, Yongqiang Tang, and Wensheng Zhang. Lorap: transformer sub-layers deserve differentiated structured compression for large language models. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 28657–28672, 2024.

Gui Ling, Ziyang Wang, Yuliang Yan, and Qingwen Liu. Slimgpt: Layer-wise structured pruning for large language models. *Advances in Neural Information Processing Systems*, 37:107112–107137, 2024.

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

Mitch Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20192–20204, 2025.

Xiang Meng, Shibal Ibrahim, Kayhan Behdin, Hussein Hazimeh, Natalia Ponomareva, and Rahul Mazumder. Osscar: one-shot structured pruning in vision and language models with combinatorial optimization. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, 2018.

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

Oliver Sieberling, Denis Kuznedelev, Eldar Kurtic, and Dan Alistarh. Evopress: Accurate dynamic model compression via evolutionary search. In *International Conference on Machine Learning*, pp. 55556–55590. PMLR, 2025.

Mingjie Sun, Zhuang Liu, Anna Bair, and Zico Kolter. A simple and effective pruning approach for large language models. In *International Conference on Learning Representations*, volume 2024, pp. 4942–4964, 2024.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Stanford alpaca: An instruction-following llama model, 2023.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

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

Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116*, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019.

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

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

A Theoretical Analysis and Derivations

A.1 The Continuous Relaxation Method

We provide a supplementary introduction to the continuous relaxation method for the optimization problem formulated in (2). Specifically, we relax the binary constraint $z \in \{0,1\}^{d_m}$ to the continuous domain $z \in [0,1]^{d_m}$, yielding the following optimization problem:

$$\min_{z \in [0,1]^{d_m}} \quad z^\top Q z \qquad \text{s.t.} \quad \mathbf{1}^\top z = k. \tag{6}$$

Since the reconstruction error matrix Q is positive semi-definite by construction, the problem in Eq. (6) is a standard convex Quadratic Programming (QP) problem, which can theoretically be solved in polynomial time (Boyd & Vandenberghe, 2004; Kozlov et al., 1979). Upon obtaining the optimal continuous solution $z^\star_{\text{relax}}$, we can derive a feasible binary solution for the original problem by projecting $z^\star_{\text{relax}}$ onto the discrete constraint set (e.g., by retaining the indices corresponding to the k largest values in $z^\star_{\text{relax}}$).

However, in the context of Large Language Models, the dimensionality of the interaction matrix Q presents a significant computational challenge. For instance, in the FFN layers of LLaMA2-7B, the intermediate dimension d_m is 11,008, which implies that Q is a square matrix of order 11,008. Solving such a large-scale Quadratic Programming problem for every single layer incurs a substantial computational cost. Therefore, in practical implementation, we adopt the more efficient Greedy Interaction Search. Nevertheless, investigating this relaxation holds theoretical value as it provides a rigorous baseline for assessing the quality of discrete algorithms.

Let $\mathcal{L}^\star_{\text{relax}}$ denote the optimal objective value of the continuous problem (6), and let $\mathcal{L}^\star_{\text{discrete}}$ be the global minimum of the original binary problem (2). Since the discrete feasible set is strictly contained within the continuous domain (i.e., $\{0,1\}^{d_m} \subset [0,1]^{d_m}$), the relationship $\mathcal{L}^\star_{\text{relax}} \leq \mathcal{L}^\star_{\text{discrete}}$ always holds. Consequently, $\mathcal{L}^\star_{\text{relax}}$ constitutes a lower bound for any valid binary solution, including the solution obtained by our proposed Greedy Interaction Search and the solution obtained by projecting the continuous relaxation. This leads to the inequality chain:

$$\mathcal{L}^\star_{\text{relax}} \leq \mathcal{L}^\star_{\text{discrete}} \leq \mathcal{L}_{\text{greedy}}, \mathcal{L}_{\text{relax}},$$

where $\mathcal{L}_{\text{greedy}}$ and $\mathcal{L}_{\text{relax}}$ represent the objective values evaluated using the feasible binary solutions derived from the Greedy Interaction Search and the projection of the continuous optimal solution, respectively.

Figure 2 illustrates the reconstruction error trends on the first FFN layer of LLaMA2-7B. We observe that our efficient Greedy Interaction Search yields reconstruction errors nearly identical to those of the projected solution, with both significantly outperforming the diagonal baseline. Furthermore, the gap between our greedy solution and the theoretical lower bound remains narrow across all sparsity levels.

A.2 Efficient Implementation and Complexity

Directly evaluating the marginal cost $\Delta_i(S)$ for all candidates at each step using (5) is computationally expensive. In a naive implementation, calculating the interaction term $\sum_{j \in S} Q_{ij}$ for all $d_m - |S|$ candidates requires $O(|S| \cdot d_m)$ operations. Over k pruning steps, this accumulates to a total complexity of $O(\sum_{t=1}^{k} t \cdot d_m) = O(k^2 d_m)$, which is prohibitive for large layers.

Here, we derive the efficient update rule used in Algorithm 1 and prove its equivalence to the naive greedy selection. We maintain an auxiliary vector $u \in \mathbb{R}^{d_m}$ initialized to zero, where $u_i^{(t)} \triangleq \sum_{j \in S_t} Q_{ij}$ represents the accumulated interaction cost at step t. When a new unit i^* is selected and added to the set (i.e., $S_{t+1} = S_t \cup \{i^*\}$), the interaction term for any remaining candidate i updates linearly:

$$\sum_{j \in S_{t+1}} Q_{ij} = \sum_{j \in S_t} Q_{ij} + Q_{i,i^*}. \tag{7}$$

This allows us to update the auxiliary vector via a simple vector addition: $u^{(t+1)} \leftarrow u^{(t)} + Q_{:,i^*}$, where $Q_{:,i^*}$ is the column of Q corresponding to the selected unit. Consequently, the marginal cost

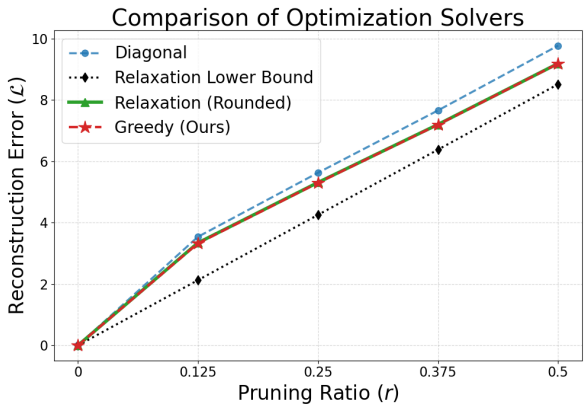

Figure 2: **Comparison of Optimization Methods.** Reconstruction error ($\mathcal{L}$) on the first FFN layer of LLaMA2-7B under varying pruning ratios (r). We plot the *Diagonal* baseline (blue), the theoretical *Relaxation Lower Bound* (black dotted), the projected *Relaxation* solution (green solid), and our *Greedy* interaction algorithm (red dashed).

evaluation $\Delta_i = Q_{ii} + 2u_i$ becomes an $O(1)$ operation, and the update step takes $O(d_m)$. This efficient implementation reduces the total complexity from quadratic $O(k^2 d_m)$ to linear $O(k \cdot d_m)$ with respect to the pruning budget k, ensuring scalability.

A.3 Justification for Block-wise Pruning

For Multi-Head Attention (MHA), pruning is typically performed at the granularity of attention heads rather than individual neurons to preserve the structural integrity of the attention mechanism. Let $\mathcal{B}_h$ be the set of channel indices corresponding to the h-th attention head, where $h \in \{1, \dots, H\}$. The decision to prune head h is represented by a binary variable $s_h \in \{0, 1\}$.

The relationship between the channel-wise mask z and the head-wise mask s is given by the constraint: $z_i = s_h$ for all $i \in \mathcal{B}_h$. Substituting this into the general reconstruction error objective $\mathcal{L}(S) = \sum_{i,j} z_i z_j Q_{ij}$, we derive the block-wise objective:

$$\begin{aligned} \mathcal{L}_{\text{head}}(s) &= \sum_{a=1}^{H} \sum_{b=1}^{H} \sum_{i \in \mathcal{B}_a} \sum_{j \in \mathcal{B}_b} z_i z_j Q_{ij} \\ &= \sum_{a=1}^{H} \sum_{b=1}^{H} s_a s_b \left(\sum_{i \in \mathcal{B}_a} \sum_{j \in \mathcal{B}_b} Q_{ij} \right). \end{aligned} \tag{8}$$

By defining the block interaction matrix $\tilde{Q} \in \mathbb{R}^{H \times H}$ with entries $\tilde{Q}_{ab} \triangleq \sum_{i \in \mathcal{B}_a} \sum_{j \in \mathcal{B}_b} Q_{ij}$, the objective simplifies to the quadratic form $s^\top \tilde{Q} s$. This derivation shows that the reconstruction objective under the block-wise constraint can be represented exactly using the aggregated matrix $\tilde{Q}$. We can therefore apply Greedy Interaction Search directly to this matrix for attention-head pruning, without introducing additional approximations in the block aggregation.

A.4 Details of Layer-Wise Sparsity Allocation

For a calibration sequence $x = (x_1, \dots, x_T)$, the end-to-end objective in Section 4.3 is

$$\mathcal{J}(x; \theta) = -\frac{1}{T-1} \sum_{t=1}^{T-1} \log p_\theta(x_{t+1} \mid x_{\le t}),$$

where $p_\theta(\cdot \mid x_{\le t})$ denotes the model's predicted next-token distribution. We backpropagate this loss through the model and accumulate the squared Frobenius norms of its gradients with respect to each

Transformer block's output over calibration sequences, obtaining an unnormalized empirical estimate of the sensitivity S_ℓ.

To convert these gradient-based sensitivity estimates into concrete pruning ratios $\{r_\ell\}_{\ell=1}^{L}$ under a fixed global sparsity level r, we employ the following normalization process, where each ℓ indexes a Transformer block. First, to handle the wide dynamic range of sensitivity values, we apply a logarithmic transformation followed by min–max normalization:

$$\tilde{S}_\ell = \log(S_\ell + \epsilon), \quad \hat{S}_\ell = \frac{\tilde{S}_\ell - \min_j \tilde{S}_j}{\max_j \tilde{S}_j - \min_j \tilde{S}_j},$$

where $\epsilon > 0$ is a small constant for numerical stability. We then convert sensitivity to a compressibility score by taking the complement of the normalized sensitivity and raising it to the power α,

$$c_\ell = (1 - \hat{S}_\ell)^\alpha,$$

where $\alpha > 0$ is a temperature parameter controlling how sharply compression is concentrated on less sensitive layers (a larger α leads to more concentrated compression). Finally, we scale the scores so that the mean sparsity matches the global sparsity level:

$$r_\ell = \frac{L c_\ell}{\sum_{j=1}^{L} c_j} r.$$

This strategy assigns lower compression rates to more sensitive layers and higher rates to less sensitive layers. Within each Transformer block, we apply the same r_ℓ to both attention heads and FFN channels; thus, the allocation varies across depth but not between the attention and FFN sublayers of the same block.

B Supplementary Experiments

B.1 Additional Experimental Settings

We prune Transformer blocks sequentially from shallow to deep layers. After pruning each block, we perform a forward pass through the pruned block using its calibration inputs and use the resulting outputs as the inputs to the next block. Thus, the activation statistics used for subsequent pruning reflect the effects of all previously pruned blocks. Within each block, we also recompute the FFN activation statistics after attention-head pruning, so that FFN-channel pruning accounts for the modified attention outputs.

For the layer-wise allocation, we set the numerical-stability constant to $\epsilon = 10^{-10}$. We control the concentration parameter α to avoid overly extreme block-wise compression and ensure that every resulting r_ℓ remains below 80%. Specifically, we use $\alpha = 1$ for global pruning ratios of 10%, 20%, and 30%, and $\alpha = 1.15$ for the higher global pruning ratio of 50%. These choices produce a relatively smooth allocation at low and moderate compression while allowing the pruning budget to concentrate more strongly on less sensitive blocks at high compression. The same r_ℓ is used for attention-head pruning and FFN-channel pruning within each block.

For downstream evaluation, we use BoolQ, PIQA, HellaSwag, WinoGrande, ARC-easy, ARC-challenge, and OpenBookQA. We follow the standard `lm-eval` protocol with its default prompt templates and benchmark-specific default metrics. Accordingly, we report standard accuracy (`acc`) for BoolQ and WinoGrande, and normalized accuracy (`acc_norm`) for PIQA, HellaSwag, ARC-easy, ARC-challenge, and OpenBookQA. For the latter tasks, normalization by continuation length mitigates the bias caused by candidate answers of different lengths and follows the standard `lm-eval` reporting convention. All experiments use FP16 and are conducted on a single GPU with 96 GB of memory.

B.2 Cumulative Contribution of Off-Diagonal Interactions

To quantify the interaction contribution neglected by independent pruning scores, we examine the cumulative off-diagonal terms along the pruning path induced by the diagonal criterion. We use the MLP down-projection layer in the first Transformer block of LLaMA2-7B and the same 128

WikiText2 calibration sequences as in the main experiments. The interaction matrix Q is constructed as defined in Section 4.1. We sort the channels by their diagonal costs, $Q_{i_1 i_1} \leq \cdots \leq Q_{i_d i_d}$, and let $S_k = \{i_1, \ldots, i_k\}$ denote the k channels selected by the independent criterion. For each candidate set, we compute the cumulative diagonal and interaction contributions as

$$D(k) = \sum_{i \in S_k} Q_{ii}, \qquad O(k) = \sum_{\substack{i,j \in S_k \\ i \neq j}} Q_{ij}, \qquad R(k) = \left| \frac{O(k)}{D(k)} \right|. \tag{9}$$

Here, $R(k)$ measures the magnitude of the cumulative interaction contribution relative to the diagonal term for the same selected channels. The interaction terms are summed with their signs preserved, and the absolute value is applied only after aggregation, allowing positive and negative interactions to cancel naturally.

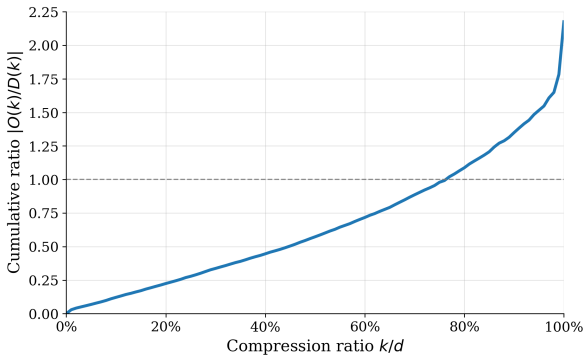

Figure 3: **Cumulative interaction-to-diagonal ratio.** Channels are ordered by ascending Q_{ii}; the curve reports $|O(k)/D(k)|$ after signed summation.

Figure 3 shows that the neglected interaction contribution increases rapidly as more channels are pruned. $R(k)$ reaches 0.337 at 30% pruning, corresponding to about one third of the diagonal contribution, increases to 0.573 at 50%, and exceeds the diagonal term at 80% with $R(k) = 1.087$. At full compression, the ratio further rises to 2.176. These results highlight the importance of accounting for off-diagonal interactions, especially at moderate and high pruning ratios.

B.3 Layer-Wise Gradient Sensitivity

We analyze the layer-wise sensitivity of LLaMA2-7B to justify our non-uniform sparsity allocation. Figure 4 reports the squared ℓ_2 norm of the gradients with respect to each layer's output. We observe a sharp decline in sensitivity from the shallow layers to the deep layers, spanning several orders of magnitude as shown in the log-scale plot. This confirms that early layers are critical for feature extraction and highly sensitive to perturbations, whereas deeper layers possess greater redundancy. Accordingly, our designed compression ratio allocation exhibits a layer-wise increasing trend.

B.4 Pruning Time

To quantify the computational overhead introduced by the pruning algorithm, we report the wall-clock pruning time on LLaMA2-7B under different pruning ratios. We use the same calibration setting as in the main experiments, i.e., 128 sequences with sequence length 2048. Specifically, we test pruning ratios of 0.1, 0.2, 0.3, and 0.5. For each setting, we repeat the pruning process 10 times and report the averaged runtime. As shown in Table 4, the pruning time is nearly insensitive to the pruning ratio. This suggests that the main runtime bottleneck lies in the calibration forward passes and interaction matrix construction, rather than in the greedy pruning algorithm itself; nevertheless, the entire pruning procedure can be completed within a few minutes, demonstrating its practical efficiency.

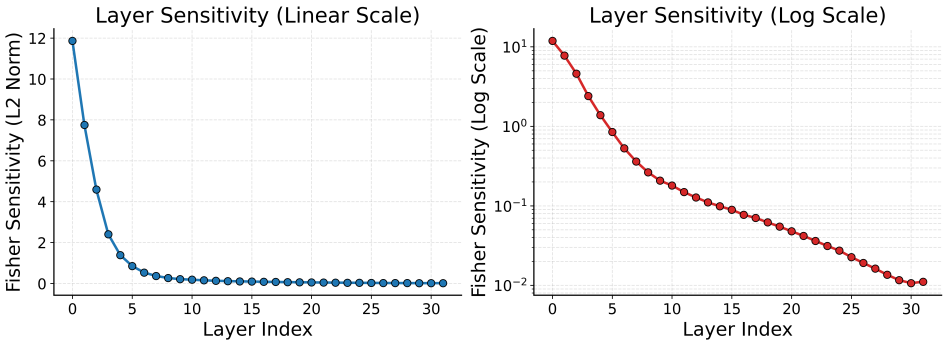

Figure 4: Layer-wise gradient sensitivity in LLaMA2-7B. We compute each layer's sensitivity as the squared ℓ_2 norm of the gradient of the next-token cross-entropy objective with respect to the layer output, and report it across depth (linear and log scales). The sensitivity generally decreases with depth, motivating an increasing pruning ratio for deeper layers.

Table 4: Average pruning time of our method on LLaMA2-7B under different pruning ratios. Each result is averaged over 10 runs.

Ratio	Time (s)
10%	184.45
20%	185.33
30%	185.35
50%	186.59

B.5 Inference Efficiency Analysis

To evaluate practical inference efficiency under comparable compression budgets, we benchmark the dense LLaMA2-7B model against physically compacted models produced by all seven pruning methods. Prefill latency is measured with a batch size of 32 and a sequence length of 128, while decoding latency covers the generation of 128 tokens for the same batch size. We perform two warm-up runs followed by 10 synchronized measurement runs and report the mean and sample standard deviation. MACs are calculated for a batch size of one and a sequence length of 128.

Table 5 reports the dense baseline and results at pruning ratios of 20% and 50%. At each pruning ratio, the similar parameter counts and MACs across methods indicate that the preceding quality comparisons are conducted under closely matched model-size budgets, ensuring a fair comparison across methods. CASP achieves the lowest prefill and decoding latencies at 20% pruning and remains in the leading latency tier at 50% pruning, together with OSSCAR and SlimGPT. These results show that CASP's structured masks translate into competitive wall-clock acceleration without relying on sparse kernels.

B.6 Performance Recovery with Fine-tuning

Here, we evaluate the performance recovery of LLaMA2-7B models pruned to 50% sparsity by CASP, LoRAP, SlimGPT, and SlimLLM. Following the LoRAP fine-tuning protocol, we train LoRA adapters on the cleaned Alpaca dataset (Taori et al., 2023) for 2 epochs with a learning rate of 1×10^{-4} and a total batch size of 64. As shown in Table 6, the results before fine-tuning are taken directly from Table 1. Fine-tuning improves all four pruning methods, while CASP retains the lowest perplexity and highest average zero-shot accuracy after recovery. In particular, its average accuracy increases from 0.4931 to 0.5343, with WikiText2 and PTB perplexities reduced to 10.68 and 47.66, respectively.

Table 5: Inference efficiency of dense and physically compacted LLaMA2-7B models. Latencies are reported in seconds as mean ± sample standard deviation over 10 synchronized runs after two warm-ups.

Ratio	Method	Params (B)	MACs (G)	Prefilling (s)	Decoding (s)
0%	Dense	6.738	850.00	0.17584 ± 0.00112	2.61359 ± 0.01692
20%	LLM-Pruner	5.423	680.75	0.15980 ± 0.00027	2.48024 ± 0.02931
	OSSCAR	5.433	681.87	0.15663 ± 0.00045	2.27846 ± 0.00542
	FLAP	5.445	684.32	0.19301 ± 0.00010	2.74126 ± 0.06009
	LoRAP	5.443	684.19	0.22944 ± 0.00008	2.42125 ± 0.00121
	SlimGPT	5.448	684.01	0.14859 ± 0.00012	2.24104 ± 0.00081
	SlimLLM	5.438	682.70	0.21785 ± 0.00009	2.61051 ± 0.01483
	CASP	5.442	683.19	0.14718 ± 0.00013	2.23260 ± 0.00106
50%	LLM-Pruner	3.508	434.42	0.14023 ± 0.00025	2.10179 ± 0.00790
	OSSCAR	3.500	433.39	0.09682 ± 0.00032	1.66766 ± 0.00041
	FLAP	3.500	433.68	0.12513 ± 0.00021	2.34689 ± 0.02418
	LoRAP	3.500	435.54	0.10895 ± 0.00006	2.09191 ± 0.00102
	SlimGPT	3.494	432.58	0.09868 ± 0.00005	1.69311 ± 0.00026
	SlimLLM	3.502	433.51	0.14228 ± 0.00008	2.30265 ± 0.01573
	CASP	3.481	430.95	0.09800 ± 0.00044	1.66767 ± 0.00120

Table 6: Performance recovery via fine-tuning on LLaMA2-7B at 50% sparsity. Results without fine-tuning match Table 1; bold values denote the best result within each setting.

Ratio	Method	WikiText2 ↓	PTB ↓	BoolQ ↑	PIQA ↑	HellaSwag ↑	WinoGrande ↑	ARC-e ↑	ARC-c ↑	OBQA ↑	Average ↑
50% w/o tune	LoRAP	23.9369	282.5295	0.6104	0.6061	0.4265	0.5549	0.3746	0.2696	0.3100	0.4503
	SlimGPT	18.3618	62.3579	0.4275	0.6322	**0.4982**	**0.6030**	**0.5210**	**0.3191**	**0.3620**	0.4804
	SlimLLM	50.7812	142.1250	**0.6303**	0.6159	0.4295	0.5462	0.4331	0.3072	0.2940	0.4652
	CASP	**15.2595**	**56.6118**	0.5924	**0.6676**	0.4949	0.5627	0.4764	0.3114	0.3460	**0.4931**
50% w/ tune	LoRAP	11.2505	131.0526	0.6471	0.6861	0.5671	0.5651	0.5008	0.2986	**0.3900**	0.5221
	SlimGPT	10.9678	52.1846	0.5569	0.6850	0.5233	0.5470	**0.5274**	0.2927	0.3660	0.4997
	SlimLLM	14.5373	61.5005	0.6306	0.6572	0.4983	0.5683	0.4756	0.2892	0.3100	0.4899
	CASP	**10.6813**	**47.6559**	**0.6725**	**0.6975**	**0.5682**	**0.6188**	0.5076	**0.3097**	0.3660	**0.5343**

B.7 Stability Across Calibration Seeds

To evaluate the stability of different pruning methods with respect to calibration sampling, we independently repeat the LLaMA2-7B experiment five times at a 50% pruning ratio and report the mean and sample standard deviation across the five runs. As shown in Table 7, CASP achieves the best mean performance on WikiText2 perplexity, PTB perplexity, and average zero-shot accuracy, while maintaining relatively small variability. These results indicate that its performance advantage is stable across repeated calibration.

Table 7: Calibration-seed stability on LLaMA2-7B at a 50% pruning ratio. Results are mean ± sample standard deviation over five calibration seeds.

Method	WikiText2 ↓	PTB ↓	Average ↑
LLM-Pruner	87.9562 ± 20.2228	345.7000 ± 86.3951	0.4145 ± 0.0047
OSSCAR	20.5967 ± 0.4550	64.7036 ± 9.5635	0.3659 ± 0.0078
FLAP	24.8719 ± 0.3212	142.0500 ± 1.8341	0.4175 ± 0.0032
LoRAP	24.3993 ± 0.2661	275.6731 ± 16.2200	0.4526 ± 0.0044
SlimGPT	18.3473 ± 0.1694	64.2272 ± 1.9676	0.4790 ± 0.0064
SlimLLM	53.8875 ± 7.6137	143.8000 ± 10.0384	0.4739 ± 0.0091
CASP	$\mathbf{15.7625} \pm 0.4512$	$\mathbf{56.0182} \pm 1.1920$	$\mathbf{0.4909} \pm 0.0053$

B.8 Additional Results on Other Models

We extend the evaluation beyond LLaMA2-7B to examine CASP on a larger model (LLaMA2-13B), Vicuna-7B, and a model with grouped-query attention (LLaMA3-8B). The following results compare language modeling performance and average zero-shot accuracy across the seven evaluation datasets.

LLaMA2-13B. Table 8 shows that CASP remains effective at a larger model scale. It achieves the lowest perplexity on both datasets and the highest average zero-shot accuracy at 20%, 30%, and 50% pruning, while remaining competitive at 10%. At 50% pruning, it obtains a WikiText2 PPL of 10.61, a PTB PPL of 49.76, and an average zero-shot accuracy of 0.5515. These results indicate consistent benefits at moderate and high pruning ratios.

Vicuna-7B. As shown in Table 9, CASP achieves the lowest perplexity on both datasets at every evaluated pruning ratio and the highest average accuracy at three of the four ratios. At 50% pruning, CASP reaches an average accuracy of 0.4900 and a WikiText2 PPL of 18.98, compared with 0.4730 and 20.21 for SlimGPT, the strongest baseline by average accuracy. The overall trend is consistent with the results on LLaMA2-7B.

LLaMA3-8B with GQA. We evaluate CASP on LLaMA3-8B to examine its applicability to grouped-query attention. We select baseline methods that can be readily adapted to GQA. Since multiple query heads share a key-value head, we treat query heads as the primary pruning units and remove a key-value head only when all associated query heads have been pruned.

As shown in Table 10, the results on LLaMA3-8B demonstrate that CASP remains effective on GQA-based models. Under the relatively low pruning ratio of 10%, CASP achieves the best perplexity on both WikiText2 and PTB, while maintaining zero-shot performance comparable to the dense model. Under the aggressive pruning ratio of 50%, CASP exhibits the strongest performance retention among the compared methods, as reflected by both lower perplexity and higher average zero-shot accuracy.

Table 8: Zero-shot performance of the compressed LLaMA2-13B. The average score is computed across seven datasets. Bold values denote the best result under the same pruning ratio.

Ratio	Method	WikiText2 ↓	PTB ↓	BoolQ ↑	PIQA ↑	HellaSwag ↑	WinoGrande ↑	ARC-e ↑	ARC-c ↑	OBQA ↑	Average ↑
0%	Ground Truth	4.8818	25.7103	0.8211	0.8030	0.7965	0.7222	0.7652	0.4881	0.4520	0.6926
10%	LLM-Pruner	6.2461	34.1562	0.7278	**0.8041**	0.7818	0.7009	0.7614	0.4770	0.4560	0.6727
	OSSCAR	5.3154	32.8850	0.8168	0.7976	0.7848	0.7269	**0.7854**	**0.5256**	0.4480	**0.6979**
	FLAP	5.3438	32.0312	0.7391	0.7922	0.7604	0.7135	0.7298	0.4428	0.4680	0.6637
	LoRAP	5.5279	30.7094	0.8306	0.8009	0.7892	**0.7301**	0.7609	0.4795	0.4640	0.6936
	SlimGPT	**5.1433**	37.4921	**0.8355**	0.7884	0.7834	0.7269	0.7786	0.4991	0.4560	0.6954
	SlimLLM	5.5781	29.5156	0.7535	0.7976	0.7818	0.7182	0.7521	0.4770	0.4340	0.6735
	CASP	5.1604	**28.5468**	0.8242	0.8003	**0.7918**	0.7182	0.7635	0.4804	**0.4700**	0.6926
20%	LLM-Pruner	8.7578	58.3438	0.6557	**0.7938**	0.7415	0.6338	0.6810	0.4334	0.4480	0.6267
	OSSCAR	5.7136	33.3645	0.8128	0.7675	0.7554	0.7174	**0.7580**	0.4744	0.4360	0.6745
	FLAP	6.1133	37.0938	0.6869	0.7612	0.7068	0.6914	0.6507	0.4010	0.4360	0.6191
	LoRAP	5.8028	37.7723	**0.8226**	0.7813	0.7551	0.7096	0.7226	**0.4804**	**0.4700**	0.6774
	SlimGPT	5.7521	36.7532	0.7761	0.7606	0.7475	0.7190	0.7382	0.4599	0.4460	0.6639
	SlimLLM	6.5781	37.9062	0.7489	0.7835	0.7518	0.7032	0.7146	0.4300	0.4340	0.6523
	CASP	**5.6672**	**30.8315**	0.8107	0.7878	**0.7741**	**0.7214**	0.7360	0.4622	0.4580	**0.6786**
30%	LLM-Pruner	11.7188	86.2500	0.5709	0.7737	0.6889	0.6212	0.6473	0.4078	0.4480	0.5940
	OSSCAR	6.6243	45.9968	0.7505	0.7443	0.7024	0.6898	0.6258	0.3908	0.4340	0.6197
	FLAP	7.1211	44.8125	0.6682	0.7247	0.6442	0.6472	0.6258	0.3908	0.3880	0.5841
	LoRAP	6.6436	52.4785	0.7344	0.7688	0.7210	0.6835	0.6931	0.4334	**0.4560**	0.6415
	SlimGPT	6.4239	52.5077	**0.7823**	0.7296	0.7141	0.6898	0.6768	0.4309	0.4280	0.6359
	SlimLLM	8.0078	49.6250	0.7242	0.7622	0.7245	**0.6946**	0.6755	0.4155	0.4480	0.6349
	CASP	**6.3597**	**34.7450**	0.7621	**0.7786**	**0.7368**	0.6851	**0.7050**	**0.4369**	0.4400	**0.6492**
50%	LLM-Pruner	55.6875	343.7500	0.3807	0.6431	0.4323	0.5454	0.3535	0.2722	0.3280	0.4222
	OSSCAR	15.6469	149.5350	0.6260	0.6159	0.4611	0.5406	0.4306	0.2696	0.3200	0.4663
	FLAP	14.2969	110.7500	0.6443	0.6621	0.5115	0.5959	0.5177	0.3191	0.3660	0.5167
	LoRAP	13.5200	207.5294	0.6755	0.6817	0.5615	0.6125	**0.5539**	0.3166	0.3340	0.5337
	SlimGPT	12.1285	100.1835	**0.6795**	0.6572	0.5751	0.6212	0.5202	0.3268	0.3480	0.5326
	SlimLLM	19.7344	96.9375	0.6535	**0.7018**	0.5932	0.6338	0.5430	**0.3541**	0.3660	0.5493
	CASP	**10.6149**	**49.7559**	0.6636	0.7002	**0.6017**	**0.6417**	0.5369	0.3166	**0.4000**	**0.5515**

Table 9: Zero-shot performance of the compressed Vicuna-7B. The average score is computed across seven datasets. Bold values denote the best result under the same pruning ratio.

Ratio	Method	WikiText2 ↓	PTB ↓	BoolQ ↑	PIQA ↑	HellaSwag ↑	WinoGrande ↑	ARC-e ↑	ARC-c ↑	OBQA ↑	Average ↑
0%	Ground Truth	6.7794	24.7264	0.8162	0.7780	0.7369	0.7064	0.7231	0.4590	0.4480	0.6668
10%	LLM-Pruner	8.6719	29.7969	0.6657	0.7677	0.7188	0.6717	0.7113	0.4283	0.4240	0.6268
	OSSCAR	7.3280	26.5768	0.7960	**0.7764**	0.7192	0.6819	0.6923	0.4403	0.4300	0.6480
	FLAP	7.6094	26.9219	0.7376	0.7530	0.7180	0.6898	0.6793	0.4215	0.4320	0.6330
	LoRAP	7.4584	27.4909	0.8006	0.7601	0.7284	0.6835	0.6835	0.4377	0.4260	0.6457
	SlimGPT	7.3965	26.6321	**0.8052**	0.7715	0.7208	**0.6906**	0.6995	0.4428	0.4300	0.6515
	SlimLLM	7.7305	28.4219	0.7725	0.7699	0.7217	0.6748	0.7062	0.4403	**0.4380**	0.6462
	CASP	**7.2343**	**25.0255**	0.8037	0.7709	**0.7301**	0.6875	**0.7235**	**0.4480**	0.4160	**0.6542**
20%	LLM-Pruner	13.1719	46.5312	0.5502	**0.7606**	0.6671	0.6117	0.6549	0.3951	0.4240	0.5805
	OSSCAR	8.8936	28.9419	0.7343	0.7573	0.6940	0.6772	0.6831	0.4164	0.4240	0.6266
	FLAP	8.8750	31.1094	0.6856	0.7307	0.6544	0.6622	0.6397	0.3788	0.3980	0.5928
	LoRAP	8.2024	31.9455	0.7554	0.7486	**0.7039**	0.6646	0.6700	0.4172	**0.4260**	0.6265
	SlimGPT	8.5607	30.4470	0.7128	0.7497	0.7032	**0.6819**	0.6822	0.4300	0.4140	0.6248
	SlimLLM	9.6406	32.3438	0.7450	0.7552	0.6837	0.6590	**0.6907**	**0.4360**	0.4240	**0.6277**
	CASP	**8.0432**	**28.4269**	**0.7572**	0.7524	0.7021	0.6567	0.6755	0.4258	0.4200	0.6271
30%	LLM-Pruner	18.6094	76.1250	0.4642	0.7205	0.6073	0.5983	0.5955	0.3660	0.3900	0.5345
	OSSCAR	9.6530	38.6641	0.6948	0.7356	0.6406	0.6606	0.6406	0.3891	0.3920	0.5933
	FLAP	11.0078	38.7812	0.6104	0.6991	0.5964	0.6314	0.6031	0.3558	0.3860	0.5546
	LoRAP	9.8260	41.4174	0.7315	0.7205	0.6729	0.6472	**0.6490**	0.3925	0.3860	0.5999
	SlimGPT	10.0075	37.9749	0.7040	0.7160	0.6468	**0.6677**	0.6460	0.3993	0.3960	0.5965
	SlimLLM	11.6719	36.3750	**0.7453**	0.7323	0.6554	0.6519	0.6351	0.4087	0.3880	0.6024
	CASP	**9.2919**	**35.0062**	0.7325	**0.7434**	**0.6738**	**0.6693**	0.6427	**0.4113**	**0.4080**	**0.6116**
50%	LLM-Pruner	122.5625	413.0000	0.5972	0.6159	0.3750	0.5209	0.3742	0.2978	0.3400	0.4459
	OSSCAR	22.6205	107.7632	0.4654	0.6284	0.4254	0.5556	0.4461	0.2654	0.3080	0.4420
	FLAP	27.6094	123.0625	0.3798	**0.6289**	0.4218	0.5588	0.4592	0.3063	0.3320	0.4410
	LoRAP	68.3384	234.1445	**0.6156**	0.6104	0.4293	0.5422	0.4032	0.2824	0.2960	0.4542
	SlimGPT	20.2140	80.8903	0.4095	0.6058	**0.4979**	**0.5919**	**0.5290**	0.3106	**0.3660**	0.4730
	SlimLLM	44.9062	140.5000	0.6013	0.6175	0.4503	0.5659	0.4571	0.3097	0.3040	0.4723
	CASP	**18.9837**	**64.6458**	0.5683	0.6251	0.4804	0.5741	0.5121	**0.3123**	0.3580	**0.4900**

Table 10: Zero-shot performance of the compressed LLaMA3-8B. The average score is computed across seven datasets. Bold values denote the best result under the same pruning ratio.

Ratio	Method	WikiText2 ↓	PTB ↓	BoolQ ↑	PIQA ↑	HellaSwag ↑	WinoGrande ↑	ARC-e ↑	ARC-c ↑	OBQA ↑	Average ↑
0%	Ground Truth	6.1367	10.5469	0.8226	0.8063	0.7922	0.7364	0.7752	0.5410	0.4480	0.7031
10%	LLM-Pruner	8.7911	13.9107	0.6881	0.7671	0.7231	0.7151	0.6881	0.4437	0.4200	0.6350
	LoRAP	7.5508	12.4922	0.7783	0.7780	0.7503	0.7017	**0.7369**	**0.5009**	**0.4240**	0.6672
	CASP	**7.1484**	**12.0156**	**0.8040**	**0.7878**	**0.7595**	**0.7261**	0.7353	0.5000	0.4140	**0.6752**
50%	LLM-Pruner	60.0247	94.5353	0.5180	0.5696	0.3351	0.5130	0.3729	0.2432	0.2900	0.4060
	LoRAP	92.8750	188.3750	0.5138	**0.6507**	**0.4229**	0.5399	0.3906	0.2594	0.2980	0.4393
	CASP	**33.2500**	**53.0312**	**0.6072**	0.6170	0.3761	**0.5414**	**0.4221**	**0.2918**	**0.3080**	**0.4520**