Heterogeneous Structured Pruning Under Resource Constraints for Hardware-Efficient Large Language Models

Cameron Barker*, School of Informatics, University of Edinburgh, UK
Andreas Paraskeva*, Leiden Institute of Advanced Computer Science, Leiden University, the Netherlands
Jan van Rijn, Leiden Institute of Advanced Computer Science, Leiden University, the Netherlands
Elliot J. Crowley, School of Engineering, University of Edinburgh, UK
Henry Gouk, School of Informatics, University of Edinburgh, UK

Abstract

Training-free pruning removes weights from pretrained large language models to reduce their deployment cost. However, existing approaches typically consider only the total number of retained weights. Consequently, their pruning decisions do not account for how target hardware stores and processes those weights. These methods select a sparsity structure in advance and apply it uniformly to all projections, rather than accounting for how each structure trades pruning error against latency and memory across sublayers. We propose a hardware-aware pruning framework that combines two components: (i) a heterogeneous search space of dense, unstructured, 2:4, block-sparse, head-pruned, and channel-pruned options and (ii) structure-regularising constraints based on latency and memory models. Our contribution generalises existing importance-based methods: a global solver selects one option per sublayer to minimise proxy pruning loss under a resource budget, thereby assigning each projection a structure and density. We evaluate the framework by pruning various LLMs (including OPT-2.7B) on an NVIDIA Jetson Orin Nano Super edge device, concluding it improves the quality–resource Pareto frontiers over the training-free baselines.

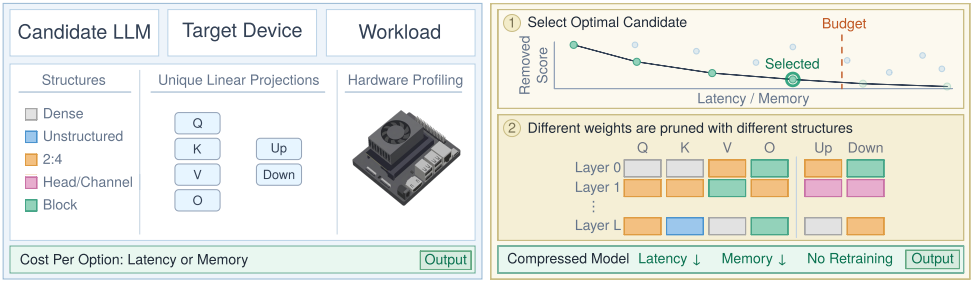


Figure 1: Our framework combines an importance metric, a resource constraint (latency or memory), and a heterogeneous search space to produce a hardware-constrained, per-sublayer pruning strategy.

*Equal contribution.
Preprint.

1 Introduction

Rapid growth of Large Language Model (LLM) sizes creates a severe deployment bottleneck for resource-constrained edge devices. Historically, the challenge of deploying convolutional neural networks (CNNs) on mobile platforms drove the development of foundational model compression techniques, such as quantisation, low-rank operators, and pruning (LeCun, Denker, and Solla 1989; Han et al. 2015; Wu et al. 2016; Han, Mao, and Dally 2016; Chollet 2017). However, simply porting these techniques to LLMs is ineffective because modern transformers introduce fundamentally different computational bottlenecks and representational structures than those found in CNNs (Zhu et al. 2024).

Training-free pruning aims to enable deployment on edge devices where one does not have the resources for retraining (M. Sun et al. 2024; An et al. 2024). While effective, most post-training pruning methods are designed to preserve accuracy under a density (model parameter count) budget, which acts as a simple hardware-agnostic proxy for resource cost. These methods lack information about the latency and memory characteristics of the target device, so hitting a density target does not always reduce resource costs. Reducing resource costs for a given target device requires careful selection of sparsity structures a priori, which may not be optimal for the specific model, hardware, and workload combination.

The choice of sparsity structure is especially challenging for LLMs due to the phase-dependent nature of transformer inference workloads: prefill is usually compute-bound, but decode is usually memory-bandwidth-bound. These two workloads require different weight sparsity structures in order to get the best performance out of the available hardware. Moreover, using the same sparsity structure across an entire model could yield suboptimal performance, as it fails to adapt to the varying computational bottlenecks of different sublayers and inference phases.

To address this gap, we propose Heterogeneous Structured Pruning (HSP), a training-free framework that jointly optimises per-sublayer sparsity structures and densities under strict resource constraints. Rather than relying on hardware-agnostic proxies such as parameter count, HSP uses profile-backed deployment costs to guide the pruning process. We show that tailoring sparsity structures on a per-sublayer basis improves the trade-off between model performance and deployment efficiency. Our main contributions are:

- **We Highlight Shortcomings of the Density Proxy:** We show experimentally that the traditional parameter density budget in existing LLM pruning methods is a poor proxy for real-world resource constraints.
- **Heterogeneous Pruning Framework:** We formulate per-sublayer structure and density selection as a global Multiple-Choice Knapsack Problem (MCKP), enabling automatic selection of multiple sparsity structures within a single model.
- **Hardware-Aware Cost Modeling:** We show that replacing parameter density budgets with hardware-aware resource models backed by workload-specific on-device latency profiles and analytical memory-footprint models results in a better trade-off between resource constraints and model quality.

2 Related Work

Post-Training Pruning: Pruning may be integrated into training or applied to an already pretrained model. Although post-training pruning (PTP) avoids pretraining a smaller model from scratch, some PTP methods still use distillation or fine-tuning to recover quality after components are removed (Muralidharan et al. 2024). Training-free methods impose various degrees of restriction. Wanda (M. Sun et al. 2024) makes no weight updates, FLAP (An et al. 2024) compensates for induced bias, while SparseGPT (Frantar and Alistarh 2023) performs a one-shot weight recovery update.

Pruning Granularity: Pruning structures span a hierarchy of granularities. At the coarsest level, layer pruning removes entire transformer layers, often using input-output similarity to identify redundancy (Men et al. 2025; Song et al. 2024). Width pruning removes coupled channels or attention heads. LLM-Pruner combines such removal with lightweight recovery tuning (Ma, Fang, and Wang 2023), whereas SliceGPT deletes rows or columns after an orthogonal transformation (Ashkboos et al. 2024). Finer-grained approaches remove weights individually, in constrained $N{:}M$ groups (Haziza et al. 2025), or in block-sparse groups (D'Alberto et al. 2024). These impose different restrictions on the admissible masks and have varying consequences for storage and execution.

Multi-Granularity Pruning: Recent methods combine several pruning granularities rather than committing to one throughout the model. Sheared LLaMA jointly prunes layers, heads, and channels during continued pretraining (Xia et al. 2024). Multi-Granularity Cascaded Pruning does PTP over the same structures in stages, using low-rank fine-tuning to recover quality (Wang et al. 2026). Neither approach considers fine-grained structures or the training-free setting.

Hardware-Aware Pruning: Hardware-aware methods optimise directly against deployment cost rather than parameter count or FLOPs. HALP formulates CNN filter pruning as a latency-constrained global knapsack problem (Shen et al. 2022). HALP uses a uniform filter structure and allocates pruning across CNN layers according to latency savings, which vary with spatial resolution. ZipLM targets runtime speedup by iteratively removing structured components with the least favourable loss-runtime trade-off, and supports both one-shot and gradual compression (Kurtić, Frantar, and Alistarh 2023). MDP jointly prunes channels, heads, and embeddings for vision workloads through a mixed-integer nonlinear program constrained by predicted latency (X. Sun et al. 2025). There is a wider field of hardware-aware neural architecture search (HW-NAS) with parallel approaches (Li et al. 2021).

In contrast, HSP performs training-free global selection over both coarse-grained structures and fine-grained subchannel pruning options for each LLM sublayer (attention or MLP). To our knowledge, this is the first work to combine heterogeneous PTP with subchannel pruning. For each workload, HSP optimises profiled latency or analytically modelled memory rather than imposing a single pruning granularity or a density proxy. This combination permits different structures and densities across the model while enforcing a deployment-resource budget.

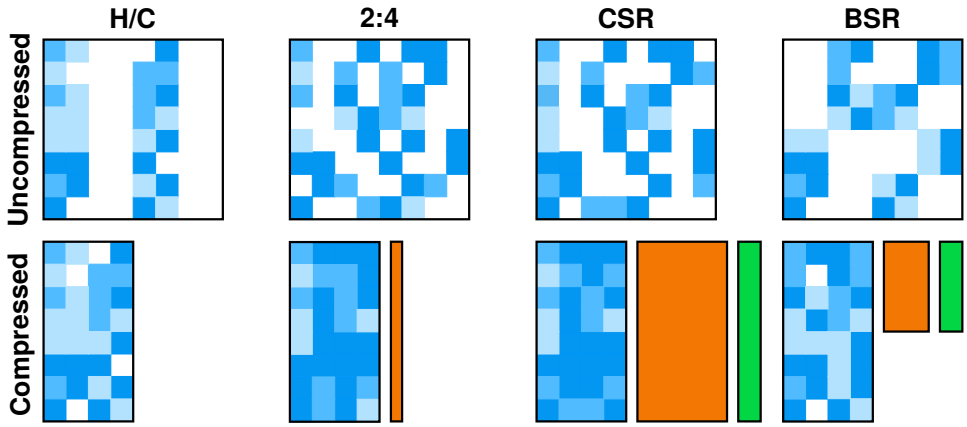


Figure 2: Compressed storage of sparse matrices with equal retained density but different structures. Retained values alone do not determine the footprint: fine-grained structures necessitate formats that store additional positional metadata. Orange and green denote different metadata components, compressed columns and row pointers respectively.

3 Pruning with Structured Sparsity

Removing weights reduces cost only when the target hardware, software kernel, and storage format can exploit the resulting sparsity. Supported structures differ across devices: coarse pruning maps naturally to smaller dense operations, whereas fine-grained sparsity may require specialised hardware primitives (e.g., NVIDIA's 2:4 sparsity) and otherwise fall back to slower general-purpose paths. Software is often the limiting factor, since hardware support provides speed-up only when exposed through efficient kernels for the relevant matrix shapes. Storage cost is similarly structure-dependent: irregular formats require more positional metadata than block-sparse formats or the compact dense matrices produced by width pruning, so equal retained density need not imply equal memory footprint (Figure 2).

3.1 Pruning Under a Density Budget

Let Π be the prunable linear projections of the model, $\mathbf{W} = \{\mathbf{W}^{(\pi)}\}_{\pi\in\Pi}$ their pretrained weights, and $\mathbf{M} = \{\mathbf{M}^{(\pi)}\}_{\pi\in\Pi}$ corresponding binary masks, where $M^{(\pi)}_{i,j} = 1$ retains a weight. Conventional density-constrained pruning seeks a mask that preserves model quality under a density budget,

$$\mathbf{M}^{\star}_{\rho} = \arg\min_{\mathbf{M}\in\mathcal{M}} \mathcal{L}(\mathbf{W}\odot\mathbf{M};\mathcal{D}) \quad \text{s.t.} \quad D(\mathbf{M}) \le \rho, \tag{1}$$

where $\mathcal{L}(\cdot\,;\mathcal{D})$ is the task loss on an evaluation set $\mathcal{D}$, $\rho \in (0,1]$ is the density target, $\mathcal{M} = \prod_{\pi\in\Pi}\mathcal{M}^{(\pi)}$ is the space of possible masks, and $D(\mathbf{M}) = \sum_{\pi\in\Pi}\|\mathbf{M}^{(\pi)}\|_0 / \sum_{\pi\in\Pi}\mathrm{numel}(\mathbf{W}^{(\pi)})$ is the retained density, with $\mathrm{numel}(\cdot)$ counting all entries of a matrix.

Importance Scores: Evaluating $\mathcal{L}$ for every candidate mask is intractable. Training-free pruning instead ranks weights using inexpensive importance scores computed from the pretrained weights and, optionally, activations collected on a small calibration set $\mathcal{D}_{\text{cal}}$. Magnitude pruning (Han et al. 2015) uses the absolute weight, whereas Wanda scales this magnitude by the norm of the corresponding input feature,

$$\mathbf{S}_{i,j}^{(\pi)\text{Mag}} = |\mathbf{W}_{i,j}^{(\pi)}|, \qquad \mathbf{S}_{i,j}^{(\pi)\text{Wanda}} = |\mathbf{W}_{i,j}^{(\pi)}| \cdot \|\mathbf{X}_{:,j}^{(\pi)}\|_2, \tag{2}$$

where $\mathbf{X}_{:,j}^{(\pi)}$ is the j-th input feature of projection π over the calibration tokens. The importance removed by a mask defines the proxy loss

$$\ell\big(\mathbf{M}^{(\pi)}, \mathbf{S}^{(\pi)}\big) = \sum_{i,j} \big(1 - \mathbf{M}_{i,j}^{(\pi)}\big)\mathbf{S}_{i,j}^{(\pi)}, \tag{3}$$

whose sum over projections, $\widetilde{\mathcal{L}}(\mathbf{M}) = \sum_{\pi\in\Pi} \ell(\mathbf{M}^{(\pi)}, \mathbf{S}^{(\pi)})$, replaces $\mathcal{L}$ in Equation 1.

Pruning Structures: The mask a method may choose is not an arbitrary binary matrix. A structure defines a *pruning unit* and a *comparison group*, and a density specifies how many units the pruning procedure retains in each comparison group. Together, a structure s and retained density d define a subset $\mathcal{M}^{(\pi)}(s,d) \subset \mathcal{M}^{(\pi)}$ of admissible masks. Unstructured pruning as used by Wanda retains a uniform number of weights in every output row, 2:4 retains two weights in every window of four consecutive input positions, fixing $d = 0.5$. For a multiweight pruning unit, we define its score as the sum of its constituent weight scores. When the comparison groups are disjoint, importance-based pruning obtains the best admissible mask by retaining the highest-scoring units in each group:

$$\mathbf{M}^{\star(\pi)}(s,d) = \operatorname*{arg\,min}_{\mathbf{M}\in\mathcal{M}^{(\pi)}(s,d)} \ell\big(\mathbf{M}, \mathbf{S}^{(\pi)}\big). \tag{4}$$

Some PTP methods such as Wanda use a uniform structure and density across all projections, whereas FLAP (An et al. 2024) selects a density per projection but still uses a uniform structure.

Coupled Structures: We treat the attention and MLP components of each decoder layer as separate *sublayers*. Each sublayer l has a coupled set of projections $\Pi_l \subset \Pi$ that share intermediate activations: $\{Q, K, V, O\}$ for standard multi-head attention and $\{U, D\}$ for a two-projection MLP. For attention, removing a head implies a mask on Q, K, and V that removes row slices of width d_{head} and a corresponding column slice of O that removes the same units. For MLPs, removing a hidden channel implies a mask on U that removes a row and a corresponding column of D. We call the projection that consumes the shared activation π_{c} (either O or D). The technical supplement describes extensions for gated MLPs and grouped-query attention.

Density Is Not a Deployment Cost: Imposing a structure restricts the admissible mask space. At fixed density, the unconstrained mask space contains every structured mask, so its minimum proxy loss cannot exceed theirs. Consequently, an objective constrained only by $D(\mathbf{M}) \le \rho$ provides no incentive to select a deployable structure, even though deployment cost depends jointly on density, structure, hardware, and the available kernel or storage implementation.

4 Heterogeneous Structured Pruning

We introduce Heterogeneous Structured Pruning (HSP), a training-free framework that introduces structure-dependent deployment cost as part of a novel bi-level optimisation formulation of pruning that allows for heterogeneous choices of sparsity structures and densities.

4.1 Resource-Constrained Bilevel Pruning

Let $\Phi^{(l)}$ be a finite set of pruning options (sparsity structure and density pairs) for sublayer l. For a given option, the inner procedure constructs the admissible mask that minimises its inner loss. The outer solver then selects an option for each sublayer that minimises the model-wide outer loss under a global resource constraint. For a model with L sublayers, this bilevel optimisation is

$$\begin{aligned} \boldsymbol{\phi}^{\star} = \underbrace{\arg\min_{\boldsymbol{\phi}\in\Phi} \sum_{l=1}^{L} \ell_{\text{out}}^{(l)}\left(\mathbf{M}^{\star(l)}\left(\phi^{(l)}\right); \phi^{(l)}\right)}_{\text{outer problem}} \\ \text{s.t.} \quad \mathbf{M}^{\star(l)}\left(\phi^{(l)}\right) = \underbrace{\arg\min_{\mathbf{M}\in\mathcal{M}^{(l)}(\phi^{(l)})} \ell_{\text{in}}^{(l)}\left(\mathbf{M}; \phi^{(l)}\right)}_{\text{inner problem}}, \\ C(\boldsymbol{\phi}) \le B. \end{aligned} \tag{5}$$

Here, $\boldsymbol{\phi} = (\phi^{(l)})_{l=1}^{L}$ is a model-wide option configuration, $\Phi = \prod_l \Phi^{(l)}$ is the set of such configurations, C is the hardware-resource cost, B is its budget, and $\ell_{\text{in}}^{(l)}$ and $\ell_{\text{out}}^{(l)}$ are the inner and outer losses for sublayer l. An option $\phi^{(l)} = (\phi^{(\pi)})_{\pi\in\Pi_l}$ assigns a structure-density choice $\phi^{(\pi)} = (s^{(\pi)}, d^{(\pi)})$ to every projection in the sublayer. These choices restrict the full sublayer mask space $\mathcal{M}^{(l)} = \prod_{\pi\in\Pi_l} \mathcal{M}^{(\pi)}$ to the admissible subset $\mathcal{M}^{(l)}(\phi^{(l)}) \subset \mathcal{M}^{(l)}$.

Importance Score Normalisation: Feature scales vary across decoder layers and projection roles, so projections with larger raw importance scores can dominate the losses. We allow the inner and outer losses to use different metrics. For either loss, let $\mathrm{S}^{(\pi)}$ denote the chosen scores for projection π. We normalise each projection by its mean:

$$\overline{S}^{(\pi)} = \frac{1}{\text{numel}(\mathbf{W}^{(\pi)})} \sum_{i,j} S_{i,j}^{(\pi)}, \qquad \widehat{S}_{i,j}^{(\pi)} = \frac{S_{i,j}^{(\pi)}}{\overline{S}^{(\pi)}}. \tag{6}$$

This makes scores comparable across projections without changing their within-projection ranking.

Inner Problem: For a fixed option, the inner problem is an otherwise standard pruning problem over its admissible masks. The inner loss and pruning procedure can therefore be instantiated by existing methods such as magnitude pruning, Wanda, or WandaSP. HSP

only requires the inner problem to return one mask $\mathbf{M}^{\star(l)}(\phi^{(l)})$ for each option. The outer problem can compare these masks independently of how they were obtained. Section 4.2 defines the option-specific mask spaces and inner losses used by HSP.

Outer Problem as MCKP: The outer loss may use any additive per-weight importance metric. The outer loss is

$$\ell_{\text{out}}^{(l)}\left(\mathbf{M}^{(l)};\phi^{(l)}\right) = w^{(l)}\left(\phi^{(l)}\right)\sum_{\pi\in\Pi_l}\ell\left(\mathbf{M}^{(\pi)},\widehat{\mathbf{S}}^{(\pi)}\right). \tag{7}$$

Here, $w^{(l)}(\phi^{(l)}) > 0$ is an option-specific weight. For each sublayer, collect the losses of the masks returned by the inner problem into

$$\boldsymbol{\ell}^{(l)} = \left(\ell_{\text{out}}^{(l)}\left(\mathbf{M}^{\star(l)}(\phi^{(l)});\phi^{(l)}\right)\right)_{\phi^{(l)}\in\Phi^{(l)}}. \tag{8}$$

Let $\mathbf{y} = (\mathbf{y}^{(l)})_{l=1}^{L}$ collect the one-hot decision vectors, where $\mathbf{y}^{(l)}$ encodes the option selected for sublayer l. The outer problem in Equation 5 is then the multiple-choice knapsack problem (MCKP)

$$\begin{aligned}\arg\min_{\mathbf{y}} \quad & \sum_{l=1}^{L}\boldsymbol{\ell}^{(l)\top}\mathbf{y}^{(l)} \\ \text{s.t.} \quad & \sum_{l=1}^{L}\mathbf{c}^{(l)\top}\mathbf{y}^{(l)} + C_0 \le B, \\ & \mathbf{1}^{\top}\mathbf{y}^{(l)} = 1, \quad \mathbf{y}^{(l)} \in \{0,1\}^{|\Phi^{(l)}|} \quad \forall l.\end{aligned} \tag{9}$$

where $\mathbf{c}^{(l)}$ contains the resource costs of the options in $\Phi^{(l)}$, and C_0 is the constant cost of the rest of the model that pruning does not touch. We solve Equation 9 using a resource-quantised Pareto-frontier dynamic program. The technical supplement gives implementation details and pseudocode.

Dominance Filtering: For an independent projection, candidate a dominates candidate b if $\ell_a \le \ell_b$ and $c_a \le c_b$, with at least one strict inequality. Replacing b with a in any independent sublayer configuration cannot increase either its additive outer loss or its resource cost, so we remove dominated candidates from each projection before constructing independent sublayer options. After forming the independent options and adding the coupled options, we apply the same test to the complete option set $\Phi^{(l)}$ for each sublayer. These filters preserve at least one optimum while reducing the MCKP choice sets (Sinha and Zoltners 1979).

4.2 Option Space

HSP combines independent per-projection options and coupled sublayer options:

$$\Phi^{(l)} = \Phi_{\text{ind}}^{(l)} \uplus \Phi_{\text{cpl}}^{(l)}. \tag{10}$$

The option type determines the inner loss:

$$\ell_{\text{in}}^{(l)}(\mathbf{M};\phi^{(l)}) = \begin{cases} \ell_{\text{ind}}^{(l)}(\mathbf{M}) & \phi^{(l)} \in \Phi_{\text{ind}}^{(l)}, \\ \ell_{\text{cpl}}^{(l)}(\mathbf{M}) & \phi^{(l)} \in \Phi_{\text{cpl}}^{(l)}. \end{cases} \tag{11}$$

Independent Options: Let $\Phi^{(\pi)}$ contain the structure-density choices for projection π, including dense, unstructured, 2:4, and block-sparse options. For an independent sublayer option, the inner procedure selects a mask for each projection separately:

$$\begin{aligned} \Phi_{\text{ind}}^{(l)} &= \prod_{\pi \in \Pi_l} \Phi^{(\pi)}, \\ \mathcal{M}^{(l)}(\phi^{(l)}) &= \prod_{\pi \in \Pi_l} \mathcal{M}^{(\pi)}(\phi^{(\pi)}). \end{aligned} \tag{12}$$

Here, $\ell_{\text{ind}}^{(l)}(\mathbf{M}) = \sum_{\pi \in \Pi_l} \ell(\mathbf{M}^{(\pi)}, \widehat{\mathbf{S}}^{(\pi)})$.

Coupled Options: For the standard attention and MLP sublayers defined above, a coupled option applies row- and column-slice structures at a shared density. The coupled option space and its admissible masks are

$$\begin{aligned} \Phi_{\text{cpl}}^{(l)} &= \left\{ (s_{\text{cpl}}^{(\pi)}, d)_{\pi \in \Pi_l} \;\middle|\; d \in \mathcal{D}_{\text{cpl}}^{(l)} \right\}, \\ \mathcal{M}^{(l)}(\phi^{(l)}) &= \left\{ \mathbf{M}^{(l)} \;\middle|\; \mathbf{M}^{(\pi)} = \left(\mathbf{M}^{(\pi_c)}\right)^{\top} \forall \pi \neq \pi_c \right\}, \end{aligned} \tag{13}$$

where $\mathbf{M}^{(\pi_c)} \in \mathcal{M}^{(\pi_c)}(\phi^{(\pi_c)})$, $s_{\text{cpl}}^{(\pi)}$ is the slice structure given to projection π by its role, and $\mathcal{D}_{\text{cpl}}^{(l)}$ is a finite set of densities. Following FLAP and WandaSP (An et al. 2024), the inner problem chooses which shared units to retain by minimising the proxy loss of the consumer projection only: $\ell_{\text{cpl}}^{(l)}(\mathbf{M}) = \ell(\mathbf{M}^{(\pi_c)}, \widehat{\mathbf{S}}^{(\pi_c)})$. The outer problem then evaluates the resulting masks across every projection in the sublayer using Equation 7.

4.3 Resource Models

We model the resource cost of a configuration $\boldsymbol{\phi}$ by summing the costs of the options selected for each sublayer and C_0,

$$C(\boldsymbol{\phi}) = C_0 + \sum_{l=1}^{L} C_l\left(\phi^{(l)}\right). \tag{14}$$

We account per sublayer rather than per projection because selecting a coupled option also changes the attention operation and the key-value cache size:

$$C_l\left(\phi^{(l)}\right) = \sum_{\pi \in \Pi_l} C_\pi\left(\phi^{(l)}\right) + \begin{cases} C_{\text{attn}}\left(H(\phi^{(l)})\right) & \text{attention}, \\ 0 & \text{MLP}, \end{cases} \tag{15}$$

where C_π is the cost of projection π under the structure and density specified by the option, and $C_{\text{attn}}\left(H(\phi^{(l)})\right)$ is the cost of the attention operation with the retained head count. For

an independent attention option, $H(\phi^{(l)})$ is the original head count because independent projection masks do not narrow the attention operation. We instantiate C as either latency, which we profile on the target device, or memory, which we compute analytically from the projection storage formats and the workload-dependent key-value cache size. For the latency objective, we profile every projection and attention operation required by the full option space on each target device. Full details are provided in the supplement.

4.4 From Solution to Pruned Model

The solver returns one option per sublayer, and turning these into a pruned model requires no further search: we apply the mask obtained by solving the inner problem. Head pruning also removes the corresponding key-value cache entries and narrows the attention operation. Following FLAP (An et al. 2024), we apply bias compensation to the consumer projection for coupled head and channel options. We apply the same correction to each projection assigned an independent block-sparse option in every experiment. For models whose projections already include bias terms, this modifies the existing biases without additional resource costs.

5 Experiments

We evaluate the performance of HSP for both resources: HSP-Lat, which is optimised under latency constraints, and HSP-Mem, which is optimised under memory constraints. We compare against the following baselines: the dense model, unstructured Magnitude and Wanda, their 2:4 variants, FLAP, and WandaSP. We evaluate HSP-Lat and HSP-Mem against the baselines using perplexity on the WikiText-2 test set (Merity et al. 2017) and zero-shot accuracy on BoolQ, PIQA, HellaSwag, WinoGrande, ARC-e, ARC-c, and OpenBookQA (C. Clark et al. 2019; Bisk et al. 2020; Zellers et al. 2019; Sakaguchi et al. 2020; P. Clark et al. 2018; Mihaylov et al. 2018). Perplexity measures retained next-token prediction quality, while zero-shot accuracy establishes whether this quality transfers to downstream language-understanding tasks. Additionally, we measure the resource usage of the selected HSP candidates and the baselines on the specific target device. We provide complementary details in the supplementary material, including: (i) evaluation-metric definitions, randomness controls, hardware setup, profiling, and kernels, (ii) analysis on calibration robustness, (iii) ablation on the impact of importance score and coupled head and channel pruning, (iv) additional results for smaller OPT models and LLaMA 7B, (v) additional results for different workloads, and (vi) wall-clock pruning and profiling costs. Generally, the device-specific profiling takes in the order of hours, but only needs to happen once per type of device, whereas the wall-clock time of the actual pruning operation is negligible (order of seconds) compared to model training.

5.1 Quality-Resource Frontiers

We first compare the quality–resource trade-offs achieved under parameter-density, latency, and memory constraints. Figure 3 presents GPU Pareto frontiers for perplexity and mean zero-shot accuracy across all three resources. Figure 4 presents the corresponding CPU

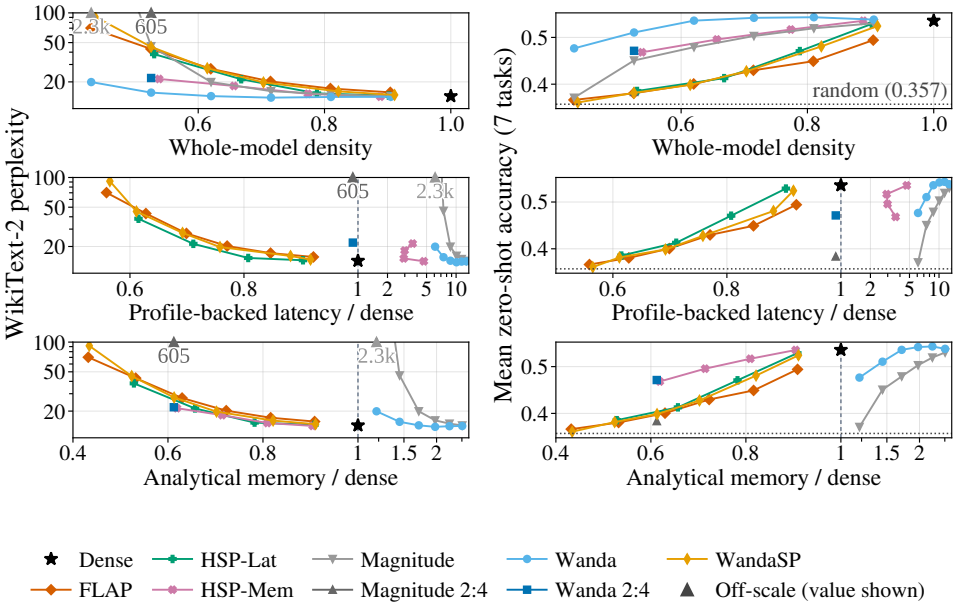


Figure 3: WikiText-2 perplexity and mean seven-task zero-shot accuracy vs. whole-model density, profile-backed latency, and analytical memory (normalised to dense) for OPT-2.7B on the Jetson Orin Nano Super GPU under the mixed workload (1024-token prefill, 128-token decode). Resource axes are linear below dense cost and logarithmic above. For candidates where perplexity exceeds the limit of 100, markers (▲) indicate the true value. The dashed rule marks random-guess accuracy (0.357).

frontiers. The quality metrics give broadly consistent method orderings. Under a density budget, unstructured magnitude and Wanda preserve the most quality per retained parameter, but have a negative impact on latency and memory savings; therefore, density-based comparisons favour candidates that are expensive to deploy.

On the GPU, HSP improves the frontier associated with its optimisation objective. HSP-Lat achieves stronger quality–latency trade-offs, whereas HSP-Mem achieves the strongest memory trade-offs. These gains do not necessarily transfer across objectives: memory-optimised candidates can have poor latency, while uniform 2:4 sparsity reduces memory without providing a comparable latency benefit. Similarly, FLAP and WandaSP reduce latency but have worse model quality than HSP at comparable cost. The pruning constraint must therefore match the target deployment bottleneck.

On the CPU, HSP-Lat again improves the quality–latency frontier. In contrast, HSP-Mem does not clearly improve the memory frontier, likely because the runtime provides fewer efficient sparse implementations, including no supported 2:4 kernel. Together, these results show that quality–resource trade-offs depend on both the chosen objective and the sparse

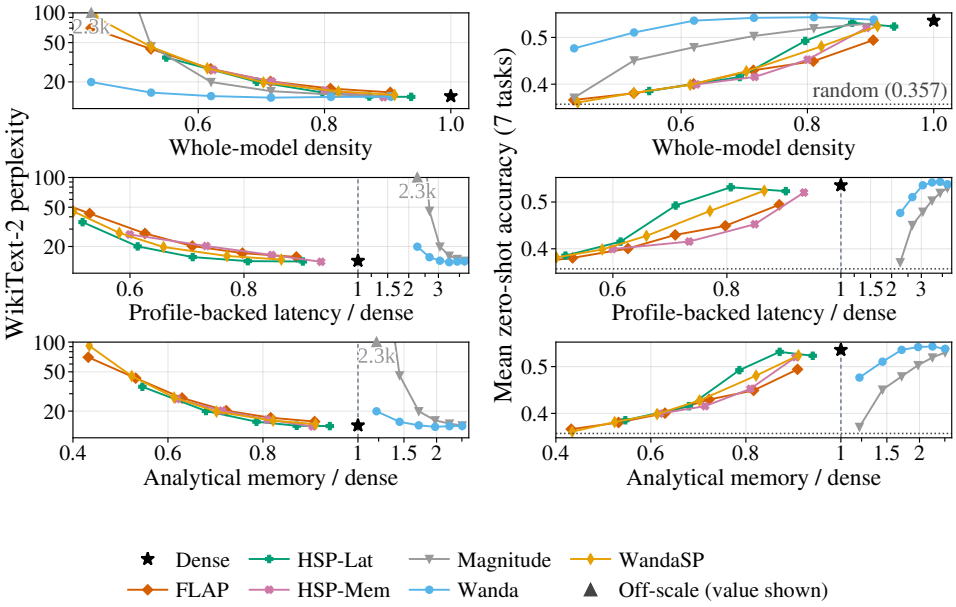


Figure 4: WikiText-2 perplexity (left) and mean seven-task zero-shot accuracy (right) vs. the same three resource axes for OPT-2.7B on the Jetson CPU, following the layout of Figure 3. The two quality metrics give consistent method orderings. 2:4 baselines are excluded because the CPU does not support 2:4 instructions.

operations supported by the target device; no single structure or density-based proxy is uniformly effective.

5.2 Structure Allocation Across Sublayers and Projections

Figure 5 shows the projection structures and densities that HSP selects for OPT-2.7B at a 0.8 budget ratio under the latency and memory objectives on the GPU and CPU. All four candidates share the same target budget hyperparameter, but the selected sublayer options are substantially different. Two patterns hold in all settings: each candidate combines multiple sparsity structures across its projections, and retained densities vary by decoder layer and projection role, even though the solver can select options that assign one uniform structure and density throughout the model. The deployment objective and device govern which structures the solver combines. On the GPU, the latency objective favours coupled head and channel pruning because narrowing the associated projections yields greater speedups with less quality loss at moderate sparsity. By contrast, the memory objective selects sublayer options that keep 90 of 192 projections dense and apply 2:4 to most of the remainder. On the CPU, where there is no hardware support for 2:4 sparsity, the selected sublayer options instead combine coupled head and channel pruning with block

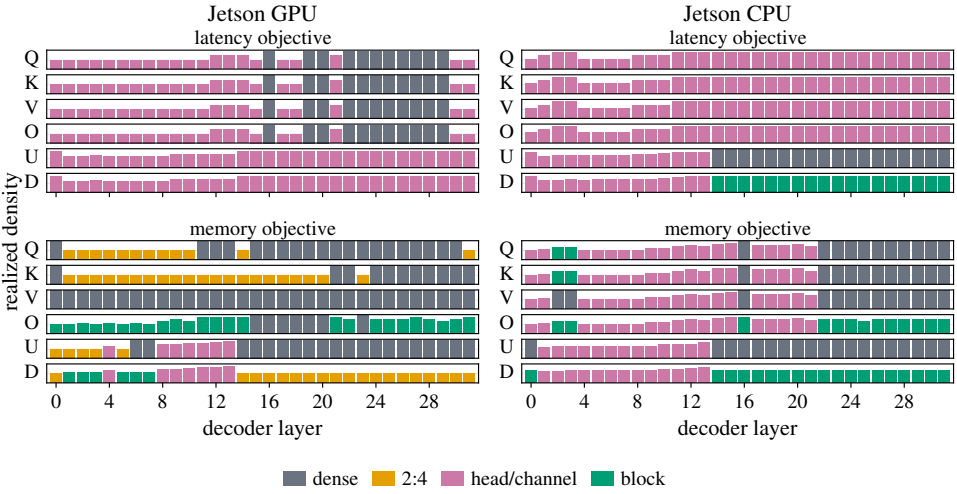


Figure 5: Structures selected by HSP for OPT-2.7B on the Jetson Orin Nano Super under the mixed workload and $B = 0.8$. Columns are the two devices and rows are the two objectives, so the panels show the GPU latency, CPU latency, GPU memory, and CPU memory objectives. Within each panel, rows correspond to the attention and MLP projections, while bars correspond to decoder layers. Bar height encodes retained density, and colour encodes the selected structure.

sparsity under both objectives and keep deeper MLP up-projections dense. These allocations demonstrate that the deployment objective and device, rather than the budget level alone, govern structure selection and validate the need for a heterogeneous search space.

5.3 On-Device Evaluation

To confirm that our offline profiles and analytical estimates capture real-world deployment dynamics, we evaluate models directly on the target device. We run OPT-2.7B on an NVIDIA Jetson Orin Nano Super at a fixed 25 W power mode, reporting on-device peak memory and end-to-end latency for different workloads (Tables 1 and 2). Unstructured candidates are excluded as they exceed memory (Section 5.1).

These results show that hardware efficiency is objective-specific. On the GPU, HSP-Lat is consistently the fastest candidate, whereas the structures selected under the memory constraint reduce storage but lead to higher latency. On the CPU, HSP-Lat provides a stronger quality–latency trade-off than FLAP and improves prefill latency over the dense model. The agreement between the predicted and measured trade-offs supports using the offline models to select pruning configurations for the deployment bottleneck of interest.

Method	PPL	Prefill		Decode		Mixed	
		Mem	Lat	Mem	Lat	Mem	Lat
Dense	14.32*			OOM			
Wanda-2:4	21.90	**3447**	0.91	**3307**	8.80	**3748**	9.62
FLAP	17.15	4519	0.64	4158	6.69	4519	9.97
HSP-Lat	15.31	4367	**0.62**	4065	**6.52**	4367	**9.12**
HSP-Mem	**15.15**	4519	11.56	4182	18.48	4621	32.86

Table 1: On-device evaluation of OPT-2.7B on the **GPU** of the Jetson Orin Nano Super. Memory (↓) is in megabytes (MB) and latency (↓) in seconds (s), measured on device. Best and second best pruned candidates are **bold** and underlined. We generate pruned candidates using a budget equal to 80% of the dense cost under its respective constraint, except Wanda-2:4. The dense model does not fit in device memory due to runtime overhead, so its perplexity is measured off-device (∗) and its on-device costs are reported as OOM. Table 2 reports the corresponding CPU results.

Method	PPL	Prefill		Decode		Mixed	
		Mem	Lat	Mem	Lat	Mem	Lat
Dense	15.66	6201	84.78	5676	27.24	6135	117.76
Wanda-2:4		No supported 2:4 kernel					
FLAP	18.23	**6512**	**66.55**	**6088**	**22.82**	**6570**	**95.54**
HSP-Lat	**15.50**	6768	67.03	6425	24.64	6748	97.16
HSP-Mem	17.68	6648	71.54	6550	22.97	6640	100.02

Table 2: On-device evaluation of OPT-2.7B on the **CPU** of the Jetson Orin Nano Super, following the units and marking conventions of Table 1. The CPU has no dedicated 2:4 hardware primitive, and software emulation was inefficient in our measurements, so our runtime provides no supported 2:4 kernel. Due to runtime constraints, CPU perplexity is measured on a 32-sample subset of WikiText-2 with 1024 tokens each, so it is not directly comparable to the GPU perplexities in Table 1.

5.4 Ablation of Search Space and Resource Proxy

In Table 3, we separately ablate the heterogeneous search space and the hardware-aware constraint to show that both are necessary. A heterogeneous search space with a density constraint achieves the best perplexity (13.73) but runs up to 113× slower than dense, showing that density is a poor proxy for on-device cost. Head and channel pruning with a latency constraint (*Single–Latency*) meets the budget but degrades quality because it restricts the solver to a narrower set of latency-efficient configurations. Our complete method (*Hetero.–Latency*) moves the quality–latency frontier by meeting the latency budget while retaining higher model quality. Neither component suffices alone: heterogeneity without a hardware-aware constraint yields the best quality at prohibitive deployment cost, but a latency constraint over a single structure meets the budget by sacrificing quality.

		Prefill		Decode		Mixed	
Search	Constraint	PPL	Lat	PPL	Lat	PPL	Lat
Dense		14.32	0.755	14.32	9.890	14.32	11.7
Single	Density	18.40	**0.525**	18.40	7.530	18.40	8.995
Hetero.	Density	**13.73**	85.5	**13.73**	31.0	**13.73**	117.5
Single	Latency	18.85	0.537	20.95	**7.045**	22.61	8.292
Hetero.	Latency	16.62	0.537	20.75	**7.045**	21.28	**8.291**

Table 3: Ablation of search space heterogeneity and constraint type for OPT-2.7B on the Jetson Orin Nano Super GPU with $B = 0.7$. *Single* uses coupled head and channel options; *Hetero.* uses the full heterogeneous search space. PPL (↓) is WikiText-2 perplexity and Lat (↓) is profile-backed latency in seconds (s). Marking follows Table 1.

6 Conclusion

In this work, we propose Heterogeneous Structured Pruning (HSP), a training-free pruning framework in which a global solver selects one option for every attention and MLP sublayer under a deployment budget, thereby assigning a sparsity structure and density to each projection. Instead of optimising a hardware-agnostic density ratio, HSP assigns every candidate structure and density an explicit deployment cost: latency obtained from one-time kernel profiles on the target device or memory computed analytically from the storage formats and the workload-dependent key-value cache. The model-wide allocation is then solved as a multiple-choice knapsack problem (MCKP). On OPT-2.7B and an NVIDIA Jetson Orin Nano Super, HSP improves the quality–resource Pareto frontier over the evaluated training-free baselines on both the CPU and GPU, as confirmed by on-device measurements of end-to-end latency and peak memory. The solver selects structurally distinct candidates under different resource constraints and on different target devices. This result demonstrates that the deployment objective and target hardware govern structure selection and that a heterogeneous search space allows a training-free method to meet latency and memory targets across hardware with seconds of search overhead.

Acknowledgements

This work was supported by the APRIL AI HUB (). Cameron Barker is supported by the UKRI EPSRC Centre for Doctoral Training in Machine Learning Systems (EP/Y03516X/1). This publication is part of the project LESSEN with project number NWA.1389.20.183 of the research program NWA ORC 2020/21 which is (partly) financed by the Dutch Research Council (NWO). This project was supported by the Royal Academy of Engineering under the Research Fellowship programme. ...

References

An, Yongqi, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. 2024. "Fluctuation-based adaptive structured pruning for large language models." In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI 2024,* 10865–10873. AAAI Press. ISBN: 978-1-57735-887-9. https://doi.org/10.1609/aaai.v38i10.28960. https://doi.org/10.1609/aaai.v38i10.28960.

Ashkboos, Saleh, Maximilian L. Croci, Marcelo Gennari Do Nascimento, Torsten Hoefler, and James Hensman. 2024. "SliceGPT: Compress Large Language Models by Deleting Rows and Columns." In *Proceedings of the International Conference on Learning Representations, ICLR 2024.* https://openreview.net/forum?id=vXxardq6db.

Bisk, Yonatan, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. "PIQA: Reasoning about Physical Commonsense in Natural Language." In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI 2020,* 7432–7439. AAAI Press.

Chollet, Francois. 2017. "Xception: Deep Learning with Depthwise Separable Convolutions." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2017,* 1800–1807. Los Alamitos, CA, USA: IEEE Computer Society. https://doi.org/10.1109/CVPR.2017.195. https://doi.ieeecomputersociety.org/10.1109/CVPR.2017.195.

Clark, Christopher, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. "BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions." In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019,* 2924–2936. Minneapolis, MN, USA: Association for Computational Linguistics. https://doi.org/10.18653/V1/N19-1300. https://doi.org/10.18653/v1/n19-1300.

Clark, Peter, 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: 1803.05457 [cs.AI]. https://arxiv.org/abs/1803.05457.

D'Alberto, Paolo, Taehee Jeong, Akshai Jain, Shreyas Manjunath, Mrinal Sarmah, Samuel Hsu, Yaswanth Raparti, and Nitesh Pipralia. 2024. *Weight Block Sparsity: Training, Compilation, and AI Engine Accelerators.* arXiv: 2407.09453 [cs.LG]. https://arxiv.org/abs/2407.09453.

Frantar, Elias, and Dan Alistarh. 2023. "SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot." In *Proceedings of the International Conference on Machine Learning, ICML 2023,* 10323–10337. PMLR. https://proceedings.mlr.press/v202/frantar23a.html.

Han, Song, Huizi Mao, and William J. Dally. 2016. "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding." In *Proceedings of the International Conference on Learning Representations, ICLR 2016,* edited by Yoshua Bengio and Yann LeCun. http://arxiv.org/abs/1510.00149.

Han, Song, Jeff Pool, John Tran, and William J. Dally. 2015. "Learning both weights and connections for efficient neural networks." In *Proceedings of the International Conference on Neural Information Processing Systems, NeurIPS 2015,* 1:1135–1143. Montreal, Canada: MIT Press.

Haziza, Daniel, Timothy Chou, Dhruv Choudhary, Luca Wehrstedt, Francisco Massa, Jiecao Yu, Geonhwa Jeong, Supriya Rao, Patrick Labatut, and Jesse Cai. 2025. *Accelerating Transformer Inference and Training with 2:4 Activation Sparsity.* arXiv: 2503.16672 [cs.LG]. https://arxiv.org/abs/2503.16672.

Kellerer, Hans, Ulrich Pferschy, and David Pisinger. 2004. *Knapsack Problems.* Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-540-24777-7. https://doi.org/10.1007/978-3-540-24777-7.

Kjolstad, Fredrik, Shoaib Kamil, Stephen Chou, David Lugato, and Saman Amarasinghe. 2017. "The tensor algebra compiler." *Proceedings of the ACM on Programming Languages, PACMPL* (New York, NY, USA) 1, no. OOPSLA (October). https://doi.org/10.1145/3133901. https://doi.org/10.1145/3133901.

Kurtić, Eldar, Elias Frantar, and Dan Alistarh. 2023. "ZipLM: Inference-Aware Structured Pruning of Language Models." In *Proceedings of the International Conference on Neural Information Processing Systems, NeurIPS 2023,* 36:65597–65617. Curran Associates, Inc. https://doi.org/10.52202/075280-2863. https://proceedings.neurips.cc/paper_files/paper/2023/hash/ced46a50befedcb884ccf0cbe8c3ad23-Abstract-Conference.html.

LeCun, Yann, John Denker, and Sara Solla. 1989. "Optimal Brain Damage." In *Proceedings of the International Conference on Neural Information Processing Systems, NeurIPS 1989,* edited by D. Touretzky, 2:598–605. Morgan-Kaufmann. https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

Li, Chaojian, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, and Yingyan (Celine) Lin. 2021. "HW-NAS-Bench: Hardware-Aware Neural Architecture Search Benchmark." In *Proceedings of the International Conference on Learning Representations, ICLR 2021.* https://openreview.net/forum?id=_0kaDkv3dVf.

Ma, Xinyin, Gongfan Fang, and Xinchao Wang. 2023. "LLM-Pruner: On the Structural Pruning of Large Language Models." In *Proceedings of the International Conference on Neural Information Processing Systems, NeurIPS 2023,* 36:21702–21720. Curran Associates, Inc. https://doi.org/10.52202/075280-0950. https://proceedings.neurips.cc/paper_files/paper/2023/hash/44956951349095f74492a5471128a7e0-Abstract-Conference.html.

Men, Xin, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2025. "ShortGPT: Layers in Large Language Models are More Redundant Than You Expect." In *Findings of the Association for Computational Linguistics, ACL 2025,* 20192–20204. Association for Computational Linguistics. https://aclanthology.org/2025.findings-acl.1035/.

Merity, Stephen, Caiming Xiong, James Bradbury, and Richard Socher. 2017. "Pointer Sentinel Mixture Models." In *Proceedings of the International Conference on Learning Representations, ICLR 2017.* https://openreview.net/forum?id=Byj72udxe.

Mihaylov, Todor, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. "Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering." In *Proceedings of the Conference on Empirical Methods in Natural Language Processing. EMNLP 2018,* 2381–2391. Association for Computational Linguistics. https://doi.org/10.18653/v1/d18-1260.

Muralidharan, Saurav, 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." In *Proceedings of the International Conference on Neural Information Processing Systems, NeurIPS 2024,* edited by A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, 37:41076–41102. Curran Associates, Inc. https://doi.org/10.52202/079017-1299. https://proceedings.neurips.cc/paper_files/paper/2024/file/4822991365c962105b1b95b1107d30e5-Paper-Conference.pdf.

Sakaguchi, Keisuke, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. "WinoGrande: An Adversarial Winograd Schema Challenge at Scale." In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI 2020,* 8732–8740. AAAI Press. https://doi.org/10.1609/AAAI.V34I05.6399. https://doi.org/10.1609/aaai.v34i05.6399.

Shen, Maying, Hongxu Yin, Pavlo Molchanov, Lei Mao, Jianna Liu, and Jose M. Alvarez. 2022. "Structural Pruning via Latency-Saliency Knapsack." In *Proceedings of the International Conference on Neural Information Processing Systems, NeurIPS 2022,* 35:12894–12908. Curran Associates Inc. https://proceedings.neurips.cc/paper_files/paper/2022/file/5434be94e82c54327bb9dcaf7fca52b6-Paper-Conference.pdf.

Sinha, Prabhakant, and Andris A. Zoltners. 1979. "The Multiple-Choice Knapsack Problem." *Operations Research* 27 (3): 503–515. ISSN: 0030364X, 15265463, accessed July 28, 2026. http://www.jstor.org/stable/170213.

Song, Jiwon, Kyungseok Oh, Taesu Kim, Hyungjun Kim, Yulhwa Kim, and Jae-Joon Kim. 2024. "SLEB: Streamlining LLMs through Redundancy Verification and Elimination of Transformer Blocks." In *Proceedings of the International Conference on Machine Learning, ICML 2024,* 235:46136–46155. PMLR. https://proceedings.mlr.press/v235/song24f.html.

Sun, Mingjie, Zhuang Liu, Anna Bair, and J. Zico Kolter. 2024. "A Simple and Effective Pruning Approach for Large Language Models." In *Proceedings of the International Conference on Learning Representations, ICLR 2024.* https://openreview.net/forum?id=PxoFut3dWW.

Sun, Xinglong, Barath Lakshmanan, Maying Shen, Shiyi Lan, Jingde Chen, and José M. Álvarez. 2025. "MDP: Multidimensional Vision Model Pruning with Latency Constraint." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2025,* 20113–20123. Computer Vision Foundation / IEEE. https://doi.org/10.1109/CVPR52734.2025.01873. https://openaccess.thecvf.com/content/CVPR2025/html/Sun%5C_MDP%5C_Multidimensional%5C_Vision%5C_Model%5C_Pruning%5C_with%5C_Latency%5C_Constraint%5C_CVPR%5C_2025%5C_paper.html.

Touvron, Hugo, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. *LLaMA: Open and Efficient Foundation Language Models.* arXiv: 2302.13971 [cs.CL]. https://arxiv.org/abs/2302.13971.

Wang, Jinghan, Yanjun Chen, Wei Zhang, Xiaotong Huang, Tianchen Liu, and Gaoliang Peng. 2026. *Cascaded Multi-Granularity Pruning for On-Device LLM Inference in Industrial IoT.* arXiv: 2606.26861 [cs.CL]. https://arxiv.org/abs/2606.26861.

Wu, Jiaxiang, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. 2016. "Quantized Convolutional Neural Networks for Mobile Devices." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016,* 4820–4828. Los Alamitos, CA, USA: IEEE Computer Society. https://openaccess.thecvf.com/content_cvpr_2016/html/Wu_Quantized_Convolutional_Neural_CVPR_2016_paper.html.

Xia, Mengzhou, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2024. "Sheared LLaMA: Accelerating Language Model Pre-training via Structured Pruning." In *Proceedings of the International Conference on Learning Representations, ICLR 2024.* https://openreview.net/forum?id=09iOdaeOzp.

Zellers, Rowan, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. "HellaSwag: Can a Machine Really Finish Your Sentence?" In *Proceedings of the Conference of the Association for Computational Linguistics, ACL 2019,* 4791–4800. Florence, Italy: Association for Computational Linguistics. https://doi.org/10.18653/V1/P19-1472. https://doi.org/10.18653/v1/p19-1472.

Zhu, Xunyu, 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, TACL* (Cambridge, MA) 12:1556–1577. https://doi.org/10.1162/tacl_a_00704. https://aclanthology.org/2024.tacl-1.85/.

A Reproducibility and Experimental Configuration

A.1 Option and Hyperparameter Configuration

Option Families and Device Support For each attention or MLP sublayer, HSP constructs independent and coupled options. An independent option assigns each projection a dense, unstructured, or rectangular-block structure; the GPU search also includes NVIDIA's fixed 2:4 pattern. A coupled option instead prunes shared attention heads across Q, K, V, and O, or shared MLP channels across U and D.

We write block shapes as output rows × input columns. The GPU search uses the Cartesian product $\{1, 16, 32, 64\} \times \{1, 16, 32, 64\}$, whereas the CPU search uses $\{1, 8, 16, 32\} \times \{1, 8, 16, 32\}$. The 1×1 case is represented by the unstructured option; every other pair is block sparse. We exclude 2:4 from the CPU search because the CPU provides no specialised hardware support for it.

Density Grid and Floors Unstructured and rectangular-block options use requested densities $\{0.1, 0.2, \ldots, 0.9\}$, restricted by a structure-specific minimum. For OPT, the minima are 0.3 for unstructured options and 0.5 for rectangular blocks, heads, and channels. The CPU memory search is an exception, which raises the rectangular-block minimum to 0.7 while leaving the other minima unchanged. The GPU 2:4 option has a fixed density of 0.5.

Coupled options enumerate every retained head count and every MLP width on the profiling grid at or above the applicable minimum. For the reported OPT and LLaMA configurations, both correspond to density increments of $1/H$, where H is the number of attention heads. The coupled minimum is 0.5 for OPT and 0.125 for LLaMA 7B.

For an unstructured option, each output row retains the nearest integer to the requested fraction of its weights. For a block-sparse option, each block-grid row similarly retains the nearest integer number of blocks. Half-integer ties are rounded upward in both cases.

Figure A.1 breaks down the rectangular-block selections in the main paper's structure-allocation figure. Maximising GPU performance requires Tensor Core execution through the mma instruction. This instruction requires block dimensions of at least 16×16, although larger blocks are typically needed in practice to approach peak throughput. Pareto filtering removes the smaller, inefficient block options from the search. The larger block options that remain use coarser pruning units and therefore remove more importance, so the GPU latency candidate selects no rectangular blocks. The GPU memory candidate can instead select smaller blocks, which reduce storage but do not provide the same latency reduction. CPU kernels saw stronger speed-ups even for smaller blocks.

Hyperparameter Development We initially imposed no structure-specific minimum densities, but importance-score proxies became unreliable at low densities. Without minimums, the global removed-importance score could be reduced by pruning a few projections or sublayers to very low densities, causing model quality to collapse. We performed a

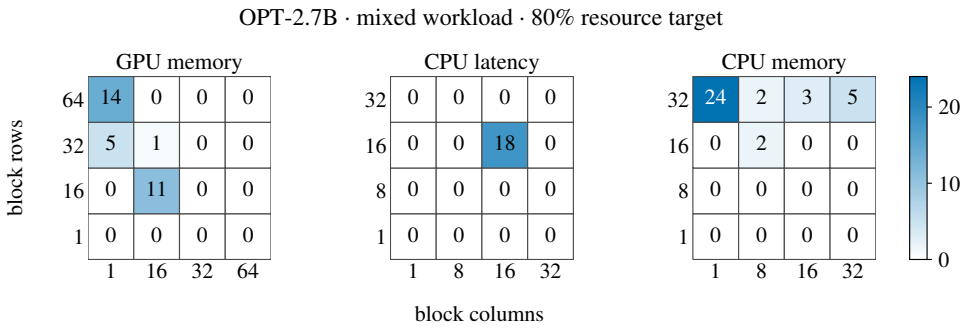


Figure A.1: Rectangular block-shape breakdown for the OPT-2.7B candidates in the main paper's structure-allocation figure under the mixed workload at $B = 0.8$. Each bar counts projection assignments, and shapes are written as output rows × input columns. The GPU-memory, CPU-latency, and CPU-memory candidates are shown; GPU latency is omitted because it selects no rectangular-block projections.

sweep of the minimums on a development model and workload using WikiText-2 validation perplexity as the quality metric. We then picked values that worked well across settings instead of using extensive per-experiment tuning.

Coupled-Option Loss Calibration We permit an option-specific weight on the outer loss of each inner-problem solution. In an early development experiment, we compared $w \in \{1.0, 0.95, 0.85, 0.7, 0.5\}$ for the coupled-option solver loss.

The value $w = 0.95$ gave a small validation-perplexity improvement for latency-constrained searches and we retained it for the main experiments.

A.2 Hardware and Compute

Edge Device We use the NVIDIA Jetson Orin Nano Super as the target deployment device for our experiments. It has a six-core ARM Cortex-A78AE CPU with NEON vectorisation and a GPU with 1024 CUDA cores and 32 Tensor Cores. The CPU and GPU share 8 GB of LPDDR5 memory with a peak bandwidth of 102.4 GB/s. We use the NVIDIA JetPack 6.2.1 SDK, which includes L4T 36.4.7, CUDA 12.6, and a prebuilt PyTorch 2.5.0 package. The Super software mode raises the available power and performance envelope on compatible Jetson Orin Nano hardware. We use its 25 W power mode for all experiments, lock the fans at 100%, and wait for the reported temperature to fall below 55 °C between profiling measurements.

Pruning Compute Pruning and search are conducted offline on a server with four NVIDIA L40 GPUs, each with 48 GB of memory. We do not use sparse kernels during pruning or search; instead, dense operations and explicit masks simulate the effect of pruning. Reported latency and memory values are obtained from the corresponding resource

models unless they are specified as "on-device" measurements, which are measured directly on the NVIDIA Jetson Orin Nano Super.

A.3 Metrics, and Randomness

Evaluation Metrics We report perplexity on the WikiText-2 test split (Merity et al. 2017). Its text rows, including empty rows, are joined with two newlines and tokenised once with the evaluated model's tokenizer. We then use 1024-token windows with stride 1024. Due to runtime constraints, the on-device CPU perplexity results in the main paper use only the first 32 windows. All other perplexity results use the full test split.

We also report the unweighted mean zero-shot accuracy across BoolQ, PIQA, HellaSwag, WinoGrande, ARC-e, ARC-c, and OpenBookQA (C. Clark et al. 2019; Bisk et al. 2020; Zellers et al. 2019; Sakaguchi et al. 2020; P. Clark et al. 2018; Mihaylov et al. 2018). We use EleutherAI's `lm-evaluation-harness` version 0.4.12 through its Hugging Face model adapter. Following the harness, we use plain accuracy for BoolQ and WinoGrande and length-normalised accuracy for the other five tasks.

The resource metrics are retained parameter density, profile-backed latency, and analytical memory including the workload-dependent key–value cache. In figures, we normalise each to the corresponding dense model unless absolute units are stated.

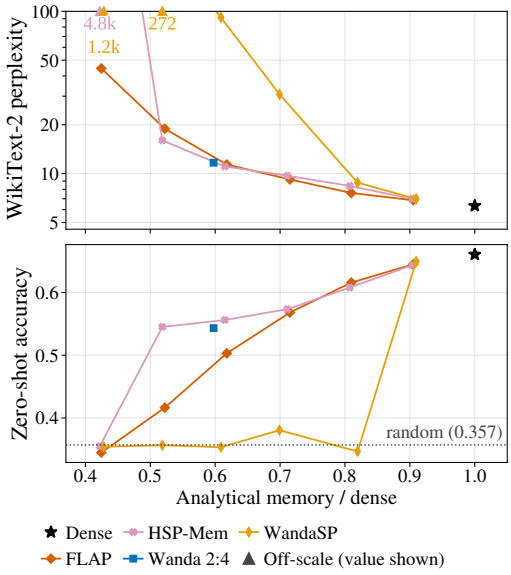
Randomness HSP is training-free and has no random initialisation. All activation-dependent pruning methods use the same 256 randomly selected windows of 1024 contiguous tokens from the WikiText-2 training split. Given these calibration statistics, mask construction, dominance filtering, and MCKP selection are deterministic. The zero-shot evaluation sets the Python, NumPy, PyTorch, and few-shot random seeds exposed by the evaluation harness to 0. Perplexity evaluation is deterministic. Kernel timing and cache control follow the measurement protocol in Section C.7.

B Scaling Across Models and Workloads

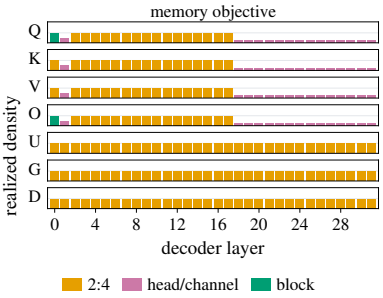
We first evaluate LLaMA 7B to test whether the main-paper comparisons extend to a larger model from a different family. We then evaluate the OPT model family across inference workloads and model scales. Throughout the supplementary scaling and workload results, a missing HSP operating point or structure panel means that HSP found no structure satisfying the corresponding resource constraint; it does not denote an unreported measurement.

B.1 LLaMA 7B

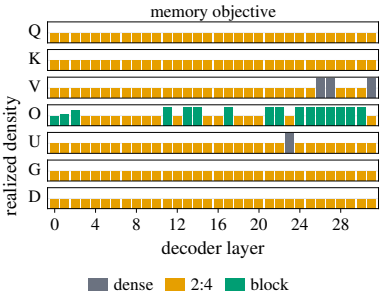
We evaluate the `huggyllama/llama-7b` checkpoint and its tokenizer in FP16 to study whether HSP scales to a larger model from a different family (Touvron et al. 2023). In FP16, this model is larger than OPT-2.7B and does not fit in the 8 GB memory shared by the NVIDIA Jetson Orin Nano. Due to the effort required to implement and tune kernels



(a) Quality–memory frontiers.



(b) Selected structures at 50% memory.



(c) Selected structures at 60% memory.

Figure B.1: LLaMA 7B using the GPU search space under the mixed workload (1024-token prefill, 128-token decode). (a) WikiText-2 perplexity (top) and mean seven-task zero-shot accuracy (bottom) versus analytical memory, normalised to the dense model; triangles mark off-scale perplexity values, which are annotated with their true values, and the dotted rule marks random-guess accuracy (0.357). (b)–(c) Structures and retained densities selected by HSP for each projection role in each decoder layer, under 50% and 60% memory constraints.

for a new edge device, we instead evaluate HSP under a memory constraint and leave latency evaluation for future work. We otherwise use the methodology from the main paper and widen the GPU search space to include head and channel densities as low as 12.5%. Figure B.1 reports the resulting frontiers and allocations. Across the plotted memory range, HSP-Mem preserves perplexity and zero-shot accuracy more consistently than FLAP (An et al. 2024). It also provides a variable-memory frontier, whereas fixed-density Wanda-2:4 (M. Sun et al. 2024) offers only a single operating point.

Selected structures Figures B.1b and B.1c show the structures selected by the solver under 50% and 60% memory constraints, respectively.

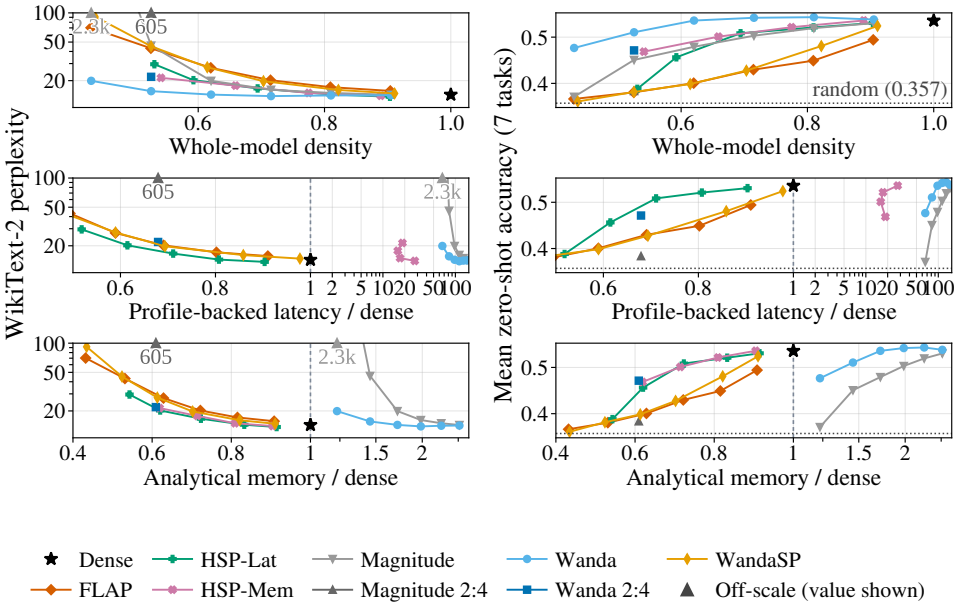


Figure B.2: WikiText-2 perplexity and mean seven-task zero-shot accuracy versus whole-model density, profile-backed latency, and analytical memory (normalised to dense) for OPT-2.7B on the Jetson Orin Nano Super GPU under the prefill-only workload (1024-token prefill). Resource axes are linear below dense cost and logarithmic above. For candidates where perplexity exceeds the limit of 100, markers (▲) indicate clipped points and annotations give the true values. The dashed rule marks random-guess accuracy (0.357).

B.2 OPT Family

Scaling Across Workloads

Figures B.2–B.5 report prefill and decode separately to test whether the conclusions drawn from the mixed workload persist for the individual inference phases. The resulting frontiers show that the quality–resource trade-offs depend on the phase, resource objective, and target processor; density alone does not provide a consistent ordering. Within each phase, the GPU figure precedes the CPU figure. The CPU figures omit 2:4 baselines because the CPU runtime provides no supported 2:4 kernel.

Scaling Across Models

Figures B.6–B.8 extend the main-paper comparison to OPT-125M, OPT-350M, and OPT-1.3B. Across model sizes, ordering candidates by retained density does not reliably predict their latency or memory ordering. The differences between the GPU and CPU frontiers further show that the quality–resource trade-offs remain target-dependent. At $B = 0.8$, HSP finds feasible allocations for ten of the twelve model–device–objective combinations across these

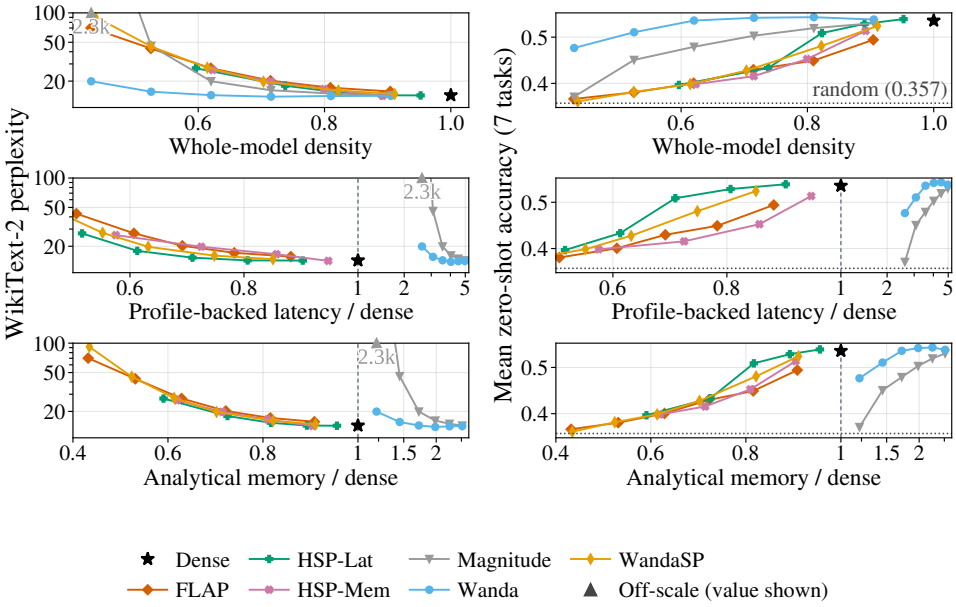


Figure B.3: WikiText-2 perplexity and mean seven-task zero-shot accuracy versus whole-model density, profile-backed latency, and analytical memory (normalised to dense) for OPT-2.7B on the Jetson Orin Nano Super CPU under the prefill-only workload (1024-token prefill). Resource axes are linear below dense cost and logarithmic above. For candidates where perplexity exceeds the limit of 100, markers (▲) indicate clipped points and annotations give the true values. The dashed rule marks random-guess accuracy (0.357).

three models; both infeasible combinations are GPU-latency targets for the two smaller models.

Selected structures Figures B.9–B.11 show the HSP allocations at $B = 0.8$. As in the main paper, the solver mixes structures and retained densities across projection roles and layers. The allocations change with model size, device, and resource objective. Only objectives for which HSP found a feasible structure are plotted. At this ratio, HSP found no structure satisfying the GPU latency constraint for OPT-125M or OPT-350M.

C Hardware and Software Support for Sparsity

C.1 Hardware Support

Different computational primitives can exploit different pruning structures. Coarse structures can be mapped to smaller parallel hardware units, while finer or irregular sparsity often falls back to slower general-purpose execution paths. NVIDIA Ampere Sparse Tensor

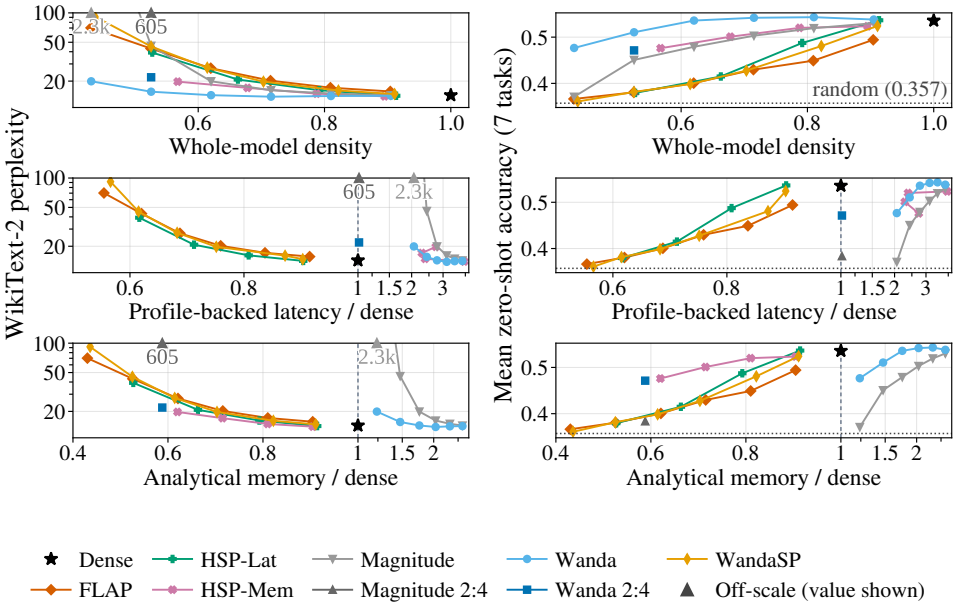


Figure B.4: WikiText-2 perplexity and mean seven-task zero-shot accuracy versus whole-model density, profile-backed latency, and analytical memory (normalised to dense) for OPT-2.7B on the Jetson Orin Nano Super GPU under the decode-only workload (128 generated tokens). Resource axes are linear below dense cost and logarithmic above. For candidates where perplexity exceeds the limit of 100, markers (▲) indicate clipped points and annotations give the true values. The dashed rule marks random-guess accuracy (0.357).

Cores found on the NVIDIA Jetson Orin Nano Super directly accelerate the fine-grained 2:4 pattern, in which each group of four entries contains two retained values. NPUs and other specialised accelerators can support different patterns, but support is often poorly documented and hidden behind proprietary compilers.

C.2 Software and Kernel Availability

Hardware support is useful only through a compatible software kernel. Historically, sparse kernels have targeted high-performance computing and scientific workloads with large matrices and high sparsity. We found no available sparse implementation that was efficient across all of the smaller operator shapes and moderate densities in our edge workloads, while sparse compilers typically prioritise flexibility and portability over raw performance (Kjolstad et al. 2017).

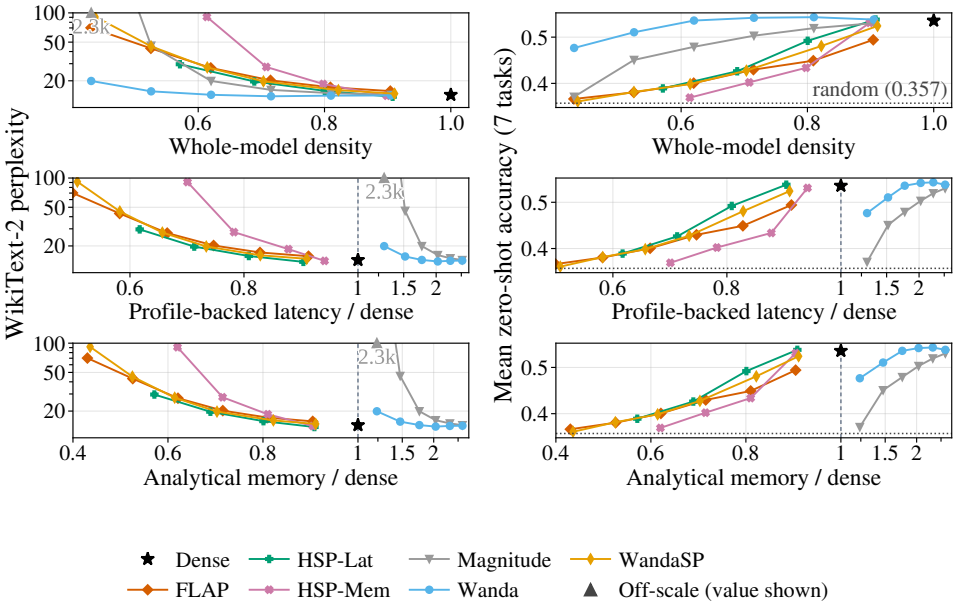


Figure B.5: WikiText-2 perplexity and mean seven-task zero-shot accuracy versus whole-model density, profile-backed latency, and analytical memory (normalised to dense) for OPT-2.7B on the Jetson Orin Nano Super CPU under the decode-only workload (128 generated tokens). Resource axes are linear below dense cost and logarithmic above. For candidates where perplexity exceeds the limit of 100, markers (▲) indicate clipped points and annotations give the true values. The dashed rule marks random-guess accuracy (0.357).

C.3 Sparse Storage Formats

A sparse format stores retained values together with the metadata needed to locate them. Our implementation represents unstructured weights in compressed sparse row (CSR) format and block-sparse weights in block sparse row (BSR) format. Each retained value or block has one column index, and every block row has an explicit row pointer. GPU 2:4 weights instead use the NV24 format, which stores the values in a dense matrix half the width of the original matrix and the column indices in a compact bit-packed format. Head and channel pruning physically narrow dense projections. Consequently, equal parameter densities can have different stored sizes.

C.4 Resource Models

Analytic Memory Model The memory objective sums the bytes stored by these deployment formats. All values are FP16 (2 bytes). For $\mathbf{W} \in \mathbb{R}^{m \times n}$, an $r \times c$ BSR encoding with z retained blocks (CSR is equivalent when $r = c = 1$), and the CUTLASS 2:4 layout, the

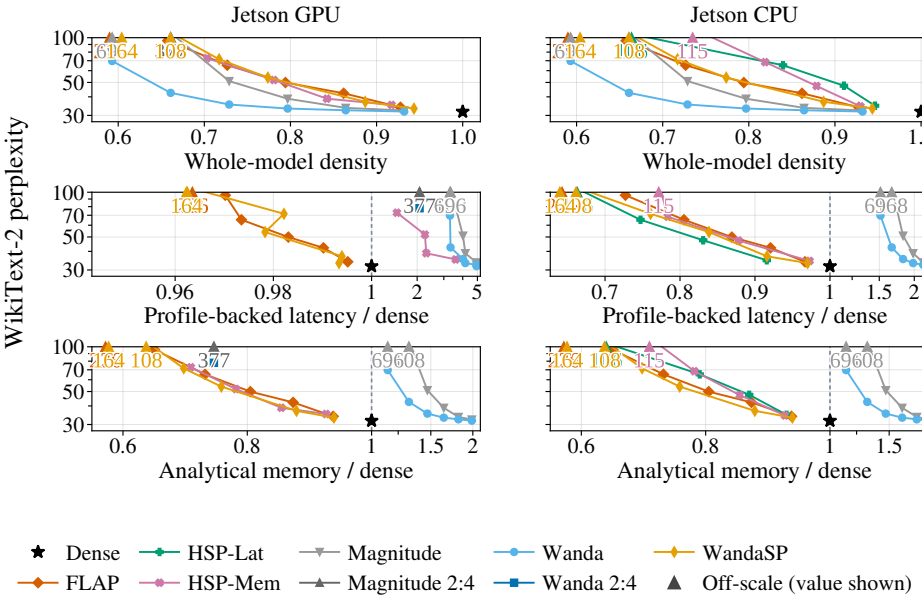


Figure B.6: WikiText-2 perplexity versus whole-model density, profile-backed latency, and analytical memory (normalised to dense) for OPT-125M on the Jetson Orin Nano Super GPU (left) and CPU (right) under the mixed workload (1024-token prefill, 128-token decode). Resource axes are linear below dense cost and logarithmic above. For candidates where perplexity exceeds the limit of 100, markers (▲) indicate clipped points and annotations give the true values.

projection costs are

$$\begin{aligned} C_{\text{dense}} &= 2mn, \\ C_{\text{BSR}} &= 2zrc + 4z + 4\left(\frac{m}{r} + 1\right), \\ C_{2:4} &= 2\frac{mn}{2} + 2\frac{mn}{16} = \frac{9mn}{8}, \end{aligned} \tag{C.1}$$

where BSR values are FP16 and its column indices and row pointers are `int32`. The unstructured case follows the same expression with $r = c = 1$. CUTLASS 2:4 stores $mn/2$ FP16 values and one `int16` metadata entry per 16 original values. For head and channel pruning, the dense expression is applied to the physically narrowed projections; retained biases are also counted in FP16.

For our batch-one workloads, let T be the final cache length (prefill plus decode tokens), and let $H_{\text{KV}}^{(\ell)}$ and $d_h^{(\ell)}$ be the retained key–value head count and head dimension in layer ℓ.

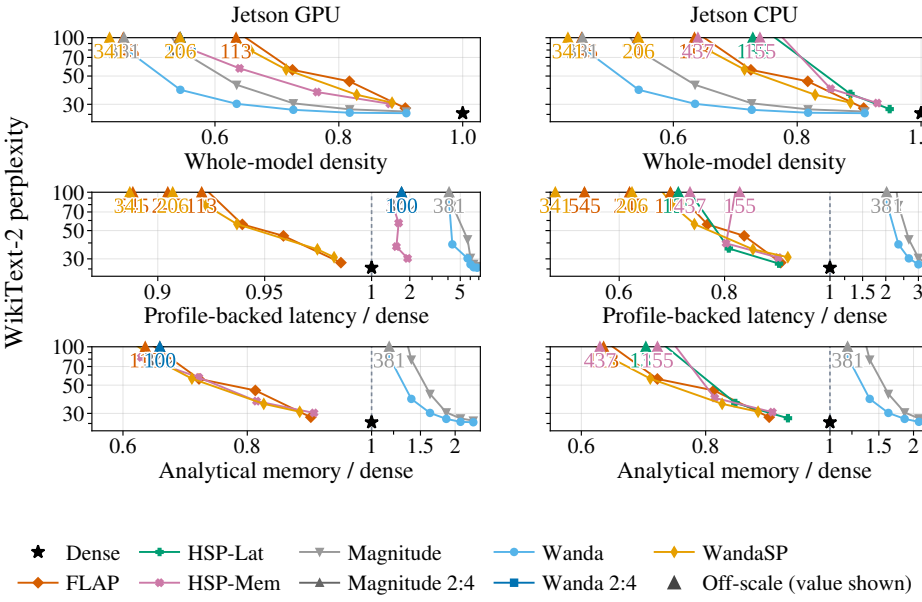


Figure B.7: WikiText-2 perplexity versus whole-model density, profile-backed latency, and analytical memory (normalised to dense) for OPT-350M on the Jetson Orin Nano Super GPU (left) and CPU (right) under the mixed workload (1024-token prefill, 128-token decode). Resource axes are linear below dense cost and logarithmic above. For candidates where perplexity exceeds the limit of 100, markers (▲) indicate clipped points and annotations give the true values.

The final FP16 key–value cache costs

$$C_{\mathrm{KV}} = 4T \sum_{\ell} H_{\mathrm{KV}}^{(\ell)} d_h^{(\ell)}, \tag{C.2}$$

where the factor 4 is two bytes each for keys and values. Whole-model memory is the sum of the projection encodings, retained non-projection parameters (including embeddings, layer norms, and biases), and this final cache. The accounting does not include temporary activations, allocator overhead, or kernel workspaces. The on-device experiments separately report measured peak memory.

Profile-Backed Latency Model For latency-constrained pruning, we assign each candidate the summed measured latencies of its induced projection and attention operations rather than using parameter count or FLOPs as a runtime proxy. The kernel pool, exact profile lookup, mixed-workload accounting, and measurement protocol are described below.

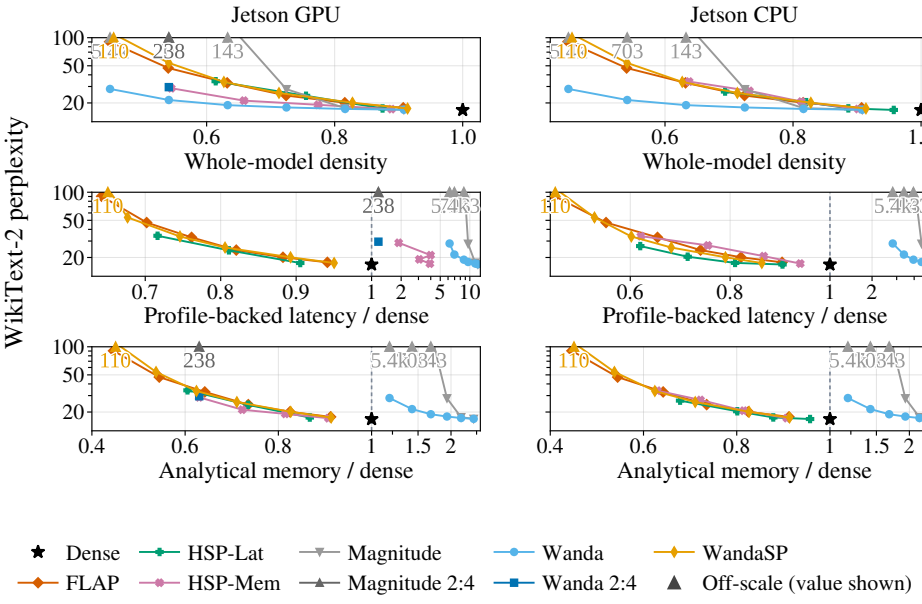


Figure B.8: WikiText-2 perplexity versus whole-model density, profile-backed latency, and analytical memory (normalised to dense) for OPT-1.3B on the Jetson Orin Nano Super GPU (left) and CPU (right) under the mixed workload (1024-token prefill, 128-token decode). Resource axes are linear below dense cost and logarithmic above. For candidates where perplexity exceeds the limit of 100, markers (▲) indicate clipped points and annotations give the true values.

C.5 Our Kernels

We develop and release a set of CUDA and CPU kernels for the NVIDIA Jetson Orin Nano Super to support this work.

Realising latency reductions from heterogeneous structured pruning depends on both hardware support and the availability of suitable high-performance kernels. Available open-source kernels and compilers did not provide sufficient performance for all structures and matrix shapes in our workloads, so we augment the available framework operators with kernels specialised for our search space.

All deployed kernels use FP16 weights and activations with FP32 accumulation. The GPU profile pool combines PyTorch dense and attention operators, a CUTLASS implementation of 2:4 sparsity, and our CUDA kernels for block-sparse operations and 2:4 decode. These paths cover both prefill and decode and use Tensor Cores when the structure and matrix dimensions permit.

For the Jetson CPU, we implement multithreaded C++ dense and block-sparse kernels that

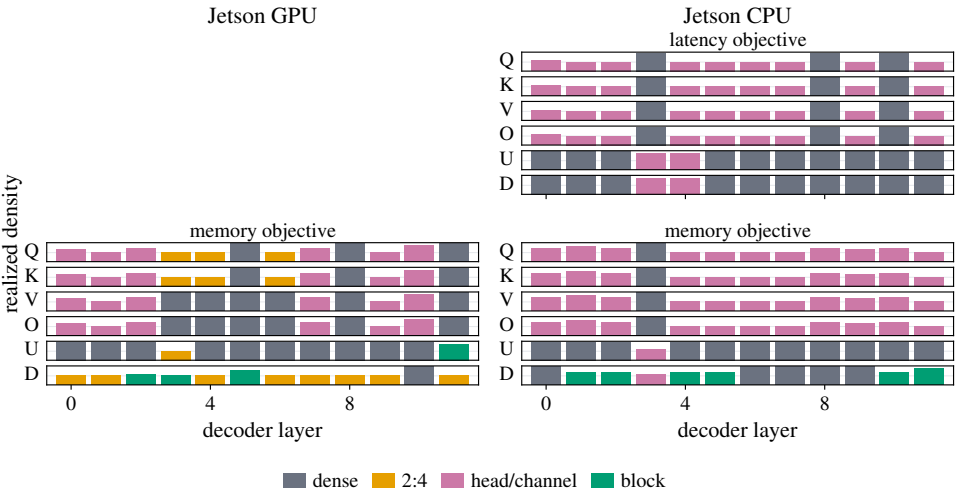


Figure B.9: Structures selected by HSP for OPT-125M on the Jetson Orin Nano Super under the mixed workload at retained-resource ratio $B = 0.8$. Columns are the two devices and rows are the two objectives; the GPU latency panel is empty because HSP found no structure satisfying that constraint at this ratio. Within each panel, rows correspond to the attention and MLP projections, while bars correspond to decoder layers. Bar height encodes retained density, and colour encodes the selected structure.

use ARM NEON SIMD instructions. PyTorch's dense FP16 CPU kernels are included in the comparison sweep, but our implementations are substantially faster for these workloads (PyTorch uses a generic implementation) and therefore form the final CPU latency profiles. The CPU runtime does not support 2:4, which was inefficient without a dedicated hardware primitive.

C.6 Kernel Profiles

We profile every projection and attention operation induced by the HSP-Lat option space on the NVIDIA Jetson Orin Nano Super in its fixed 25 W power mode. For each operation, the sweep measures compatible implementations and a curated set of launch configurations using one fixed set of random FP16 inputs, weights, and structure-compatible masks per profiling job, then retains the implementation with the lowest median latency.

Each induced operation is matched exactly by device, inference phase, matrix dimensions, sparsity structure, and retained density where applicable. We profile full coverage of the search space and do not interpolate or extrapolate missing entries. For a mixed workload, prefill is counted once and decode once per generated token.

Figures C.1–C.4 show representative projection profiles across structures, retained densities,

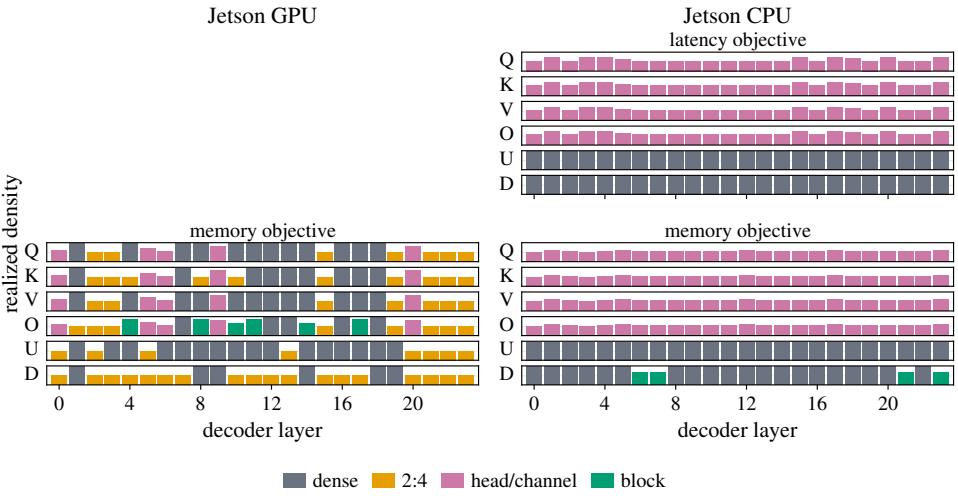


Figure B.10: Structures selected by HSP for OPT-350M on the Jetson Orin Nano Super under the mixed workload at retained-resource ratio $B = 0.8$, following the layout of Figure B.9. The GPU latency panel is again empty because HSP found no structure satisfying that constraint at this ratio.

model sizes, and devices. We report linear-scaling efficiency

$$E_{\text{lin}}(d) = \frac{d\, T_{\text{dense}}}{T_{\text{sparse}}}. \tag{C.3}$$

Here, d is retained density and T is measured latency. Expressed as a percentage, 100% denotes ideal inverse-density scaling, while values above $100d$% indicate a speed-up over dense execution.

Coupled head or channel pruning generally gives the highest GPU efficiency, although the largest block kernels catch up for D at high prefill densities. At 50% density, 2:4 and the best block kernel cross dense break-even as projections grow—by OPT-1.3B for O and OPT-350M for D. The decode D profile shows a different crossover: channel pruning remains strongest, while 2:4 and the fastest block kernel beat dense at 50% density but the block kernels fall below break-even at high densities. On the CPU O projection, head pruning and several block shapes approach linear scaling, whereas unstructured sparsity remains below dense break-even.

Statically optimising kernels to a fixed post-pruning pattern could improve efficiency, particularly for unstructured and block sparsity. We do not consider such model-specific optimisation, so the measured efficiencies are not upper bounds.

Kernel Profile Sweeps

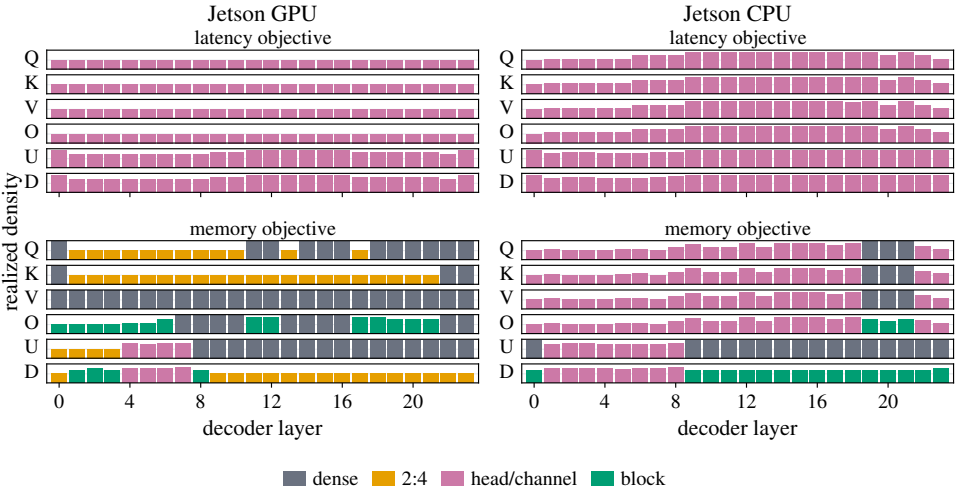


Figure B.11: Structures selected by HSP for OPT-1.3B on the Jetson Orin Nano Super under the mixed workload at retained-resource ratio $B = 0.8$, following the layout of Figure B.9. All four device and objective combinations are feasible at this ratio.

C.7 Measurement Protocol

All profiles and on-device evaluations use the fixed power, fan, and thermal controls described above. For each kernel configuration, we perform two warm-ups followed by ten timed repetitions and use the median latency. We wait for the device temperature to fall below 55 °C and flush the L2 cache before each iteration. GPU timings use CUDA events and CPU timings use a monotonic high-resolution clock.

For whole-model results, we perform one warm-up followed by five timed GPU executions or three timed CPU executions and report the median. Peak memory is measured separately using allocator peak memory on the GPU and peak process resident set size on the CPU. Across the twelve HSP-Lat and HSP-Mem device–workload pairs in the on-device table in the main paper, the median absolute relative error, $|L_{\text{meas}}/L_{\text{prof}} - 1|$, between summed profile latency and measured end-to-end latency is 2.8%. Ten of the twelve measurements are within 6% of their profile estimates; the largest difference is 18.3% for GPU decode, for which the profile sum overestimates the measured latency.

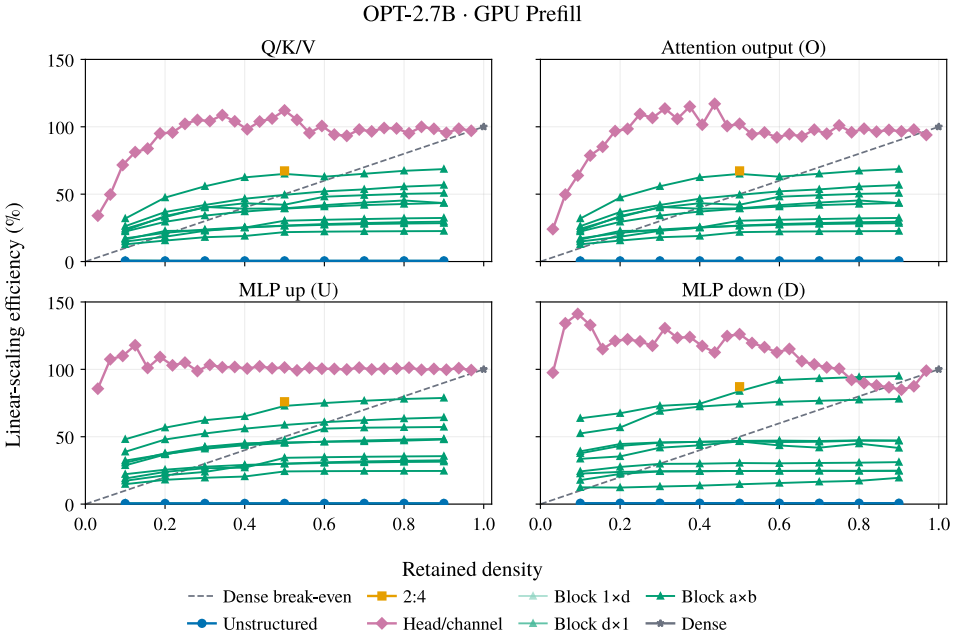


Figure C.1: Linear-scaling efficiency across retained densities for OPT-2.7B projection kernels during GPU prefill. Panels show the Q/K/V, attention output (O), MLP up (U), and MLP down (D) projections. Each exact rectangular block shape is plotted separately and grouped into a block family in the legend. Points above the dashed $100d\%$ line are faster than dense execution; 100% denotes linear inverse-density scaling.

Model	Orin Nano CPU		Orin Nano GPU	
	Configs	Time (min)	Configs	Time (min)
OPT-125M	2,004	10.4	2,964	4.7
OPT-350M	2,104	18.5	3,064	7.3
OPT-1.3B	2,464	77.4	3,424	25.8
OPT-2.7B	2,464	110.6	3,424	39.0

Table C.1: One-time kernel-profiling cost per model on the Jetson Orin Nano. "Configs" is the number of measured kernel configurations, and time is their total measurement wall-clock time.

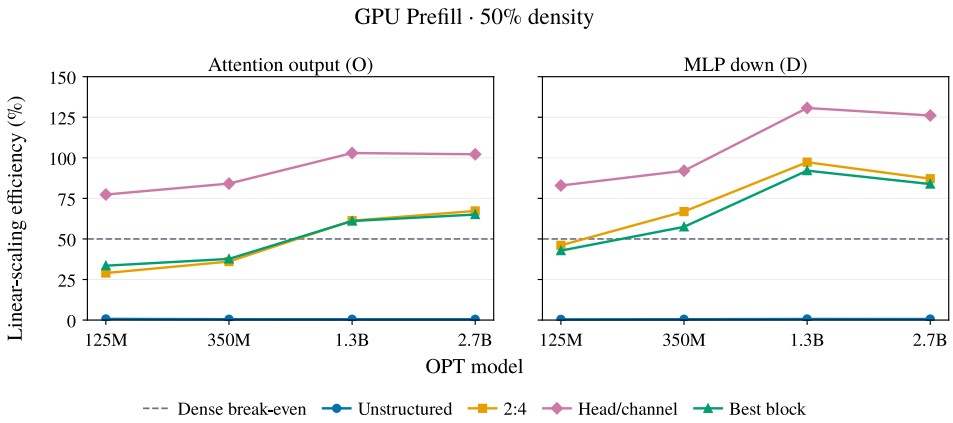


Figure C.2: GPU-prefill linear-scaling efficiency at retained density $d = 0.5$ across the OPT model family for the attention output (O) and MLP down (D) projections. "Best block" is the fastest profiled rectangular block shape for each model; this is 64×64 in every case shown. The dashed 50% line marks break-even with dense execution.

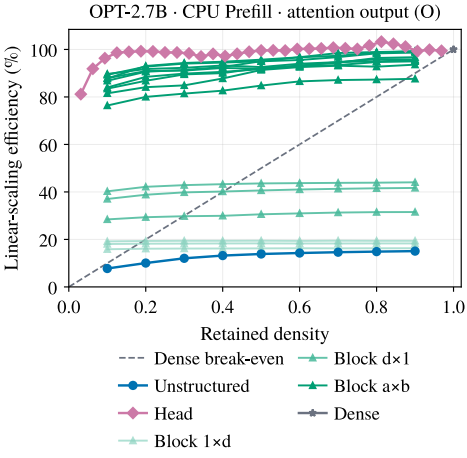


Figure C.3: Linear-scaling efficiency across retained densities for the OPT-2.7B attention output (O) projection during CPU prefill. The CPU profiles exclude 2:4, for which the CPU has no dedicated hardware primitive. Points above the dashed $100d\%$ line are faster than dense execution.

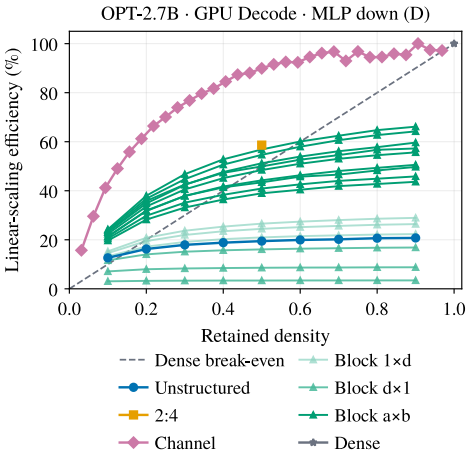


Figure C.4: Linear-scaling efficiency across retained densities for the OPT-2.7B MLP down (D) projection during one-token GPU decode. Points above the dashed $100d\%$ line are faster than dense execution.

D HSP Procedure and Candidate Realisation

D.1 HSP Algorithm

Algorithm 1 Heterogeneous Structured Pruning

Require: Weights $\mathbf{W}$; calibration set $\mathcal{D}_{\text{cal}}$; option sets $\{\Phi^{(l)}\}_{l=1}^{L}$; resource cost C; budget B
Ensure: Configuration $\boldsymbol{\phi}^{\star}$ and pruned weights $\mathbf{W}$

Compute normalised importance scores from $\mathbf{W}$ and $\mathcal{D}_{\text{cal}}$
for $l = 1$ **to** L **do**
    Solve the inner problem for each $\phi^{(l)} \in \Phi^{(l)}$ and collect its mask $\mathbf{M}^{\star(l)}(\phi^{(l)})$, loss $\boldsymbol{\ell}^{(l)}$, and cost $\mathbf{c}^{(l)}$
    $(\Phi^{(l)}, \boldsymbol{\ell}^{(l)}, \mathbf{c}^{(l)}) \leftarrow \text{ParetoFilter}(\Phi^{(l)}, \boldsymbol{\ell}^{(l)}, \mathbf{c}^{(l)})$
end for
MCKP-DP: $\mathcal{F} \leftarrow \{(\varnothing, 0, 0)\}$
for $l = 1$ **to** L **do**
    $\mathcal{F} \leftarrow \text{Extend}(\mathcal{F}, \Phi^{(l)}, \boldsymbol{\ell}^{(l)}, \mathbf{c}^{(l)})$
    $\mathcal{F} \leftarrow \text{ParetoFilter}_{\Delta}(\mathcal{F}; B - C_0)$
end for
$\boldsymbol{\phi}^{\star} \leftarrow \text{LowestLoss}(\mathcal{F})$
$\mathbf{W} \leftarrow \text{Realise}\left(\mathbf{W}, \{\mathbf{M}^{\star(l)}(\phi^{\star(l)})\}_{l=1}^{L}\right)$
Apply bias compensation **return** $\mathbf{W}, \boldsymbol{\phi}^{\star}$

D.2 MCKP Solver

For the final allocation, we follow the standard dynamic-programming approach to MCKP (Kellerer, Pferschy, and Pisinger 2004), using resource quantisation. In Algorithm 1, $\mathcal{F}$ stores partial configurations and their cumulative loss and cost, and Δ is the resource bin width. Our NumPy-vectorised implementation extends the frontier with every option for a sublayer in parallel array operations. As Table D.1 shows, the solver takes only 0.39–1.93 s across the reported configurations; the complete pruning decision takes 7.99–22.05 s.

D.3 Coupled-Structure Extensions

The main paper defines coupled masks for two-projection MLPs and standard multi-head attention. Here we give the corresponding extensions for gated MLPs and grouped-query attention (GQA).

Gated MLPs A gated MLP adds a gate projection G alongside the up projection U. We apply the same retained-channel row mask to U and G and the corresponding column mask to the down projection D; thus, $\mathbf{M}_G = \mathbf{M}_U$. The inner loss remains the consumer-projection loss on D, while the option cost includes U, G, and D.

Grouped-Query Attention In GQA, query and key–value projections have different numbers of heads, so the standard multi-head-attention transpose relation does not apply directly. Let H_Q query heads share H_{KV} key–value heads, and let $g : \{1, \ldots, H_Q\} \to \{1, \ldots, H_{\mathrm{KV}}\}$ map each query head to its key–value head. Let $\mathcal{R}_Q \subseteq \{1, \ldots, H_Q\}$ be the indices of the retained query heads and let $g(\mathcal{R}_Q) = \{g(h) : h \in \mathcal{R}_Q\}$ be the key–value heads that they use. Let $\mathbf{M}^{(l)}(\mathcal{R}_Q, g(\mathcal{R}_Q))$ retain the row slices of Q and the corresponding column slices of O for $\mathcal{R}_Q$, and the row slices of K and V for $g(\mathcal{R}_Q)$. For a coupled option $\phi^{(l)}$ specifying retained query and key–value densities d_Q and d_{KV}, respectively, the admissible mask space is

$$\begin{aligned}\mathcal{M}^{(l)}(\phi^{(l)}) = \big\{\mathbf{M}^{(l)}(\mathcal{R}_Q, g(\mathcal{R}_Q)) : \mathcal{R}_Q \subseteq \{1, \ldots, H_Q\},\\ |\mathcal{R}_Q| = d_Q H_Q, \quad |g(\mathcal{R}_Q)| = d_{\mathrm{KV}} H_{\mathrm{KV}}\big\}.\end{aligned} \tag{D.1}$$

Thus, query heads can be pruned individually, while a key–value head is removed exactly when none of its query heads belongs to $\mathcal{R}_Q$. This GQA construction is an extension of the option space; it is not evaluated in this work. Additional constraints on Equation D.1 may be preferable for efficient fixed-shape kernels, for example requiring a uniform number of retained query heads per retained key–value head, or pruning complete key–value groups rather than individual query heads.

D.4 Bias Compensation

Consider a projection $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$ with mask $\mathbf{M}$. Let $\bar{\mathbf{x}}$ be the mean of the rows of the input-activation matrix $\mathbf{X}^{(\pi)}$ over the calibration tokens. We use the compensated bias

$$\widetilde{\mathbf{b}} = \mathbf{b} + (\mathbf{W} - \mathbf{W} \odot \mathbf{M})\,\bar{\mathbf{x}}, \tag{D.2}$$

which matches the original projection output at the calibration mean. For HSP, we apply Equation D.2 to the consumer projection (O or D) of each coupled option and to every independently block-sparse projection, using that projection's input mean. We do not compensate coupled producer projections or independent unstructured and 2:4 masks. All reported OPT HSP runs enable this correction and modify their existing biases, so no additional storage or latency is introduced. The bias-free LLaMA-7B HSP runs disable bias compensation.

D.5 Pruning Wall-Clock Time

Generating a complete pruning decision for OPT-2.7B takes 8–22 s with HSP, compared with 0.33–1.15 s for the baselines (Table D.1). The primary overhead is the construction of solutions to the inner problem for every sublayer or projection option. MCKP solving and mask realisation are minor contributors. Overall, HSP completes in seconds and avoids the hours of parameter-efficient recovery tuning or billions of tokens required by retraining methods (Ma, Fang, and Wang 2023; Xia et al. 2024). Device profiling is a one-time cost (1.8 h on the CPU and 0.7 h on the GPU) that can be reused across budgets, workloads, importance metrics, and models with the same profiled projection and attention-operation shapes (Table C.1). The development and tuning costs of the kernels were unmeasured, but

Method	Time (s)			
	Options	Solve	Select	Total
Wanda	–	–	–	1.15
FLAP	–	–	–	0.33
HSP-Lat (CPU)	19.22	1.93	0.89	22.05
HSP-Lat (GPU)	20.30	0.39	0.31	21.01
HSP-Mem (CPU)	4.16	0.86	3.98	9.00
HSP-Mem (GPU)	5.44	0.79	1.75	7.99

Table D.1: Time to produce an OPT-2.7B pruning decision (↓), using precomputed calibration statistics. "Options" constructs the inner-problem solutions, "Solve" measures the MCKP runtime, and "Select" covers mask construction and bias compensation.

constituted a significant engineering effort. This effort can be amortised across future work, and we release our kernels to support further research in this area. We also emphasise that improved compilers and kernel libraries in this area are a promising avenue for future work.

E Calibration Robustness

We vary the calibration sample count for latency-constrained OPT-2.7B on the GPU under the mixed workload at $B = 0.8$, holding all other settings fixed.

Calibration Samples	4	16	64	256
Zero Input-Feature Norms (%)	6.44	4.83	4.52	4.46
Perplexity (↓)	16.947	15.129	14.880	15.299

Table E.1: Calibration sample-count sensitivity. The first row is the percentage of recorded projection-input features with zero accumulated ℓ_2 norm.

As Table E.1 shows, four samples leave more input features unobserved and give worse perplexity. From 16 to 256 samples, both quantities are relatively stable and perplexity does not improve monotonically. We use 256 samples for the main experiments rather than tuning the calibration count to this sweep.

F Importance Score and Structure Ablation

Table F.1 shows that tight latency budgets expose weaknesses in both the importance metric and the available structures. Magnitude alone ignores input activations and becomes unreliable when substantial pruning is required, whereas Wanda remains markedly more stable across workloads. Removing the coupled head and channel options narrows the

Workload	Method	Latency budget			
		0.6	0.7	0.8	0.9
Prefill	HSP (Wanda)	**20.22**	**16.62**	**14.41**	**13.65**
	HSP (Magnitude)	1.49k	539.36	47.16	15.05
	HSP (Wanda, no H/C)	–	18.26	15.07	13.69
Decode	HSP (Wanda)	**39.21**	**20.75**	**16.14**	**14.15**
	HSP (Magnitude)	16.58k	6.15k	5.41k	43.61
	HSP (Wanda, no H/C)	–	–	–	17.19
Mixed	HSP (Wanda)	**38.20**	**21.28**	**15.30**	**14.51**
	HSP (Magnitude)	16.33k	4.86k	562.13	84.72
	HSP (Wanda, no H/C)	–	–	–	16.70

Table F.1: Effect of the importance score and coupled head and channel (H/C) options on perplexity for latency-constrained OPT-2.7B on the Jetson Orin Nano Super GPU across retained-resource ratios $B = 0.6$–0.9. Best and second-best values within each workload and ratio are bold and underlined, respectively; "–" means that the restricted search space contains no candidate satisfying the corresponding budget.

achievable latency range, making most tight decode and mixed settings infeasible. Their smaller effect at looser prefill budgets suggests that these options are most valuable when the hardware constraint leaves little flexibility.

G Notation Reference

This compact reference covers the notation shared by the main paper and this supplement. Superscript (π) identifies a projection and superscript (l) identifies a sublayer unless stated otherwise.

Symbol	Meaning
π, Π, Π_l	Projection index, all prunable projections, and projections in sublayer l.
l, L	Sublayer index and number of sublayers. The cache model uses ℓ for a decoder-layer index.
$Q, K, V, O;\ U, G, D$	Query, key, value, attention-output; MLP up, gate, and down projections.
$\mathbf{W}^{(\pi)}, \mathbf{X}^{(\pi)}$	Projection weights and their calibration-token input activations.
$\mathbf{S}^{(\pi)}, \widehat{\mathbf{S}}^{(\pi)}$	Raw and projection-normalised importance scores.
$\mathbf{M}^{(\pi)}, \mathcal{M}^{(\pi)}$	Binary pruning mask and its admissible mask space.
$s^{(\pi)}, d^{(\pi)}, \phi^{(\pi)}$	Sparsity structure, retained density, and their projection-level pair.
$\phi^{(l)}, \Phi^{(l)}, \boldsymbol{\phi}$	Sublayer option, its finite option set, and a model-wide configuration.
$\ell, \ell_{\text{in}}^{(l)}, \ell_{\text{out}}^{(l)}$	Removed-importance proxy loss and the sublayer inner and outer losses.
$\boldsymbol{\ell}^{(l)}, \mathbf{c}^{(l)}, \mathbf{y}^{(l)}$	Option-loss vector, option-cost vector, and one-hot option decision.
C, C_l, C_0, B	Total resource cost, sublayer cost, fixed non-prunable cost, and resource budget; reported budgets are normalised to dense where stated.
$\mathcal{D}, \mathcal{D}_{\text{cal}}, \rho$	Evaluation data, calibration data, and conventional retained-density budget.
$\boldsymbol{\phi}^{\star}, \Delta, \mathcal{F}$	Selected configuration; resource-bin width; dynamic-programming frontier.
m, n, r, c, z	Matrix dimensions, block dimensions, and retained-block count.
T, H_{KV}, d_h	Cache length, retained key–value heads, and head dimension.