Knowledge Offloading: Decomposing LLMs into Sparse Backbones and Memory Modules

Anonymous ACL submission

Abstract

LLMs encode both general capabilities and domain-specific knowledge in a single set of parameters. We ask whether this capacity can be reorganized: keeping broadly useful computation in a shared backbone, while moving specialized knowledge into external memory modules. We propose *knowledge offloading* (KOFF), a framework for decomposing a pretrained LLM into a sparse shared backbone and domain-specific memories. Starting from a frozen base model, we jointly learn a structured pruning mask and lightweight recovery modules, implemented as LoRA adapters and learned key-value caches. Across Llama and Qwen models from 3B to 8B, we find that non-trivial capacity can be moved out of the shared backbone without a large loss in model ability. At around 12% global sparsity, KOFF preserves much of the unpruned model's performance, while pruning the same frozen model without memories degrades sharply. Ablations show that LoRA and learned KV memories are complementary, and specialization analyses suggest that the learned decomposition is meaningful: language-specific neurons are preferentially removed while language-general neurons largely remain in the backbone. These results suggest that knowledge can be reallocated between a shared core and swappable external memories.

1 Introduction

Large language models (LLMs) are expected to perform in diverse settings: a single model may need to serve users in multiple languages and reason across different domains. These use cases place high demands on model capacity. Some knowledge is broadly useful, supporting general reasoning and linguistic competence, while other knowledge is highly specialized, relevant only to particular topics or languages. Despite this diversity, LLMs lack explicit mechanisms for managing how capacity is allocated across different types of knowledge.

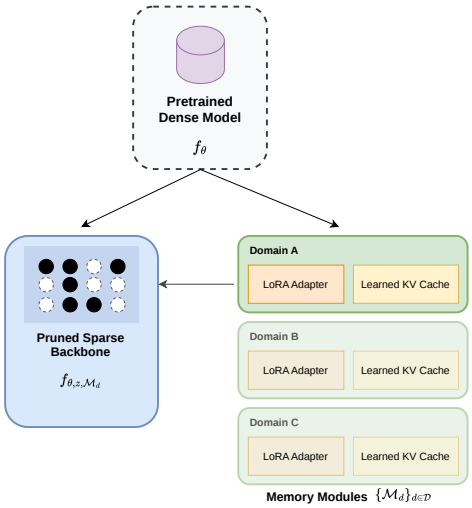

Figure 1: Knowledge offloading (KOFF) decomposes a pretrained LLM into a sparse backbone and separate swappable memory modules.

All information, both general and highly specialized, is stored within a single set of parameters. Yet, previous work shows that different subsets of neurons specialize in different functions, including language- and domain-specific processing (Tang et al., 2024; Hendy et al., 2022; Foroutan et al., 2022), suggesting that only a fraction of parameters is relevant for a given input. This raises a fundamental question: can we reorganize model capacity so that broadly useful knowledge remains in a shared core, while specialized knowledge is moved into external modules?

A natural approach is to augment LLMs with external memory. Prior work has explored retrieval-augmented generation (Lewis et al., 2020), learned KV caches (Eyuboglu et al., 2025), and parameter-efficient adapters (Hu et al., 2021) as mechanisms to extend model capacity. These approaches allow specialized knowledge to be stored outside the main parameters and accessed when needed, reducing the burden on the shared model. However, such methods typically assume a fixed back-

bone and do not decide what should remain in the model and what should be externalized. As a result, they address how to add memory, but not how to relocate capacity between shared computation and specialized knowledge. In contrast, structured pruning methods provide a way to reshape model capacity by removing parameters deemed less important (Ma et al., 2023). Yet, pruning methods are designed to produce a single compressed network that serves all inputs, forcing specialized knowledge from different domains to compete for limited capacity.

In this work, we propose *Knowledge Offloading* (KOFF), a framework that unifies memory-augmented modeling and structured pruning by learning how to relocate capacity between shared computation and external memory. Instead of compressing a model into a single smaller network, KOFF decomposes a pretrained LLM into a general-purpose backbone and domain-specific memory modules. It jointly learns two components with distinct roles: (1) a shared sparse backbone that retains broadly useful information, and (2) lightweight *memory modules* that store and recover domain-specific knowledge.

Concretely, we use a structured pruning mechanism based on Hard Concrete gates (Louizos et al., 2018) to determine which neurons are retained in the shared backbone. Simultaneously, memory modules are trained to compensate for the removed capacity through two recovery mechanisms: (a) a LoRA adapter (Hu et al., 2022) that operates in parameter space, and (b) a learnable key-value (KV) cache injected into the attention mechanism (Eyuboglu et al., 2025), acting as an externalized knowledge store. By training these components jointly, the model can reorganize capacity in a coordinated way: the backbone retains shared functionality, while the memory modules specialize in what has been offloaded. At inference time, we use a small trained classifier to select the memory module to attach to the backbone for a given input.

We evaluate knowledge offloading on Llama and Qwen models from 3B to 8B, using both topic and language domains. Our experiments show that capacity can be moved out of the shared backbone without a large loss in model ability: at around 12% global sparsity, offloading preserves much of the unpruned model's performance, while pruning the same frozen model without memory modules degrades sharply. Ablations show that LoRA adapters and learned KV caches are complementary. Finally, specialization analyses suggest that the learned decomposition is meaningful internally: language-specific neurons are preferentially removed while language-general neurons are protected, and the backbone preserves the original representation geometry. To our knowledge, this is the first work to treat pruning as a mechanism for learning memory allocation.

2 Related Work

Memory-augmented models. Previous works extend LLM capacity by externalizing knowledge in activation or parameter space. Prefix and prompt tuning inject learnable representations into the model's computation (Li and Liang, 2021; Lester et al., 2021), effectively storing task-specific knowledge in activation space rather than the base model parameters. Complementary approaches such as LoRA (Hu et al., 2022) instead introduce lightweight parameter updates, enabling models to store task-specific knowledge in parameter space. We draw inspiration from these lines of work by letting the model learn how domain-specific knowledge can be offloaded into such external memory. Closest to our work are parametric memory approaches such as memory networks (Sukhbaatar et al., 2015) and hierarchical parametric memory (Pouransari et al., 2025), which learn structured memory components alongside the model or from scratch. We instead start from a pretrained dense model, prune it, and use memory modules to recover domain-specific knowledge removed during pruning. Unlike prior work focused on adaptation of frozen models, our approach treats memory as the recovery mechanism in a co-adapted prune-and-restore system: LoRA adapters provide parameter-space memory, while KV caches act as activation-space memory.

Model pruning. Recent work has shown that LLMs can be compressed via structured pruning approaches that are applied post-training. Popular methods such as SliceGPT (Ashkboos et al., 2024), FASP (Hu et al., 2025), and LLM-Pruner (Ma et al., 2023) remove entire neurons or channels while retaining most of the base model's performance. The latter also popularized the prune-then-adapt paradigm, where a pruned model is subsequently recovered using parameter-efficient fine-tuning methods such as LoRA. However, these approaches produce a single model with a fixed quality–efficiency

trade-off. We instead use structured pruning as a mechanism for reorganizing model capacity. By learning a shared sparse backbone together with domain-specific memory modules, we allow different pruned knowledge to be restored for different inputs. This reframes pruning from a purely compressive operation into a component of a broader memory allocation strategy.

Conditional computation and expert models. Our work is also related to dense-to-MoE conversion and expert-LM methods, including MoEfication (Zhang et al., 2022), upcycling (Komatsuzaki et al., 2023), Branch-Train-Merge (BTM) (Li et al., 2022) and Branch-Train-MiX (BTX) (Sukhbaatar et al., 2024). These methods introduce conditional specialization by constructing, training or routing expert computation. In contrast, our objective involving pruning makes the allocation of capacity itself the object of learning: starting from a pretrained dense model, we learn what should remain in a shared backbone and what can be externalized into domain-specific memory modules. Architecturally, our modules are not routed FFN experts, but memory components (KV caches or LoRA) attached to a shared backbone.

3 Methodology

3.1 General framework

KOFF relies on a separation of roles. The shared backbone should preserve computation that is broadly useful across domains, while domain modules provide capacity for information specialized to a particular domain. We learn this separation by training the pruning mask jointly with the domain modules. This gives the pruning objective access to the same recovery mechanisms that will be available after pruning: capacity that can be restored by a memory module need not be retained in the shared backbone, while capacity that is difficult to recover, or useful across many domains, should remain shared. Each domain module $\mathcal{M}_d$ contains two complementary forms of memory: a parameter-space component, implemented with LoRA, and an activation-space component, implemented as a learned key-value memory. These components are described in the next subsection and are both shown beneficial in ablations (Section 4.4).

Learning objective. Let f_θ be the original pretrained model, with frozen parameters θ. We learn a shared structured pruning mask z, together with domain modules $\{\mathcal{M}_d\}_{d\in\mathcal{D}}$. The mask z is shared across domains, while each module $\mathcal{M}_d$ is domain-specific. We write $f_{\theta,z}$ for the masked backbone, and $f_{\theta,z,\mathcal{M}_d}$ for the model obtained by attaching the module for domain d.

Our goal is retention: the offloaded model should preserve the behavior of the original model while moving recoverable capacity into external memory. We therefore use distillation from f_θ, rather than standard next-token cross-entropy, both on domain data and on general retention data. At a high level, the objective is

$$\min_{z,\{\mathcal{M}_d\}} \sum_{d\in\mathcal{D}} \mathbb{E}_{x\sim\mathcal{D}_d}\left[\mathcal{L}_{\text{distill}}(f_{\theta,z,\mathcal{M}_d}, f_\theta; x)\right] + \lambda\,\Omega(z) + \mu\,\mathcal{L}_{\text{retain}}(z,\{\mathcal{M}_d\}). \tag{1}$$

The first term trains each domain module to recover the teacher's behavior on its own domain data, while updating the same shared mask z. The penalty $\Omega(z)$ encourages the shared backbone to use fewer active units; in practice, we instantiate it as an expected L_0 penalty on structured Hard Concrete gates (Section 3.3).

The retention term applies the same distillation loss on general-domain data:

$$\mathcal{L}_{\text{retain}}(z,\{\mathcal{M}_d\}) = \mathbb{E}_{x\sim\mathcal{D}_{\text{ret}},\,\tilde{d}\sim\text{Unif}(\mathcal{D})}\Big[\mathcal{L}_{\text{distill}}(f_{\theta,z,\mathcal{M}_{\tilde{d}}}, f_\theta; x)\Big]. \tag{2}$$

This discourages the shared mask from removing capacity needed outside the domain-module training distribution. In other words, domain batches teach the modules what can be recovered locally, while retention batches push the shared backbone to preserve capabilities that should remain broadly available.

Inference. The masked backbone need not be served as a dynamically gated model. After training, z is fixed and materialized: pruned neurons and their associated weight rows and columns are removed, yielding a compact dense backbone. At inference time, the domain label is not assumed to be known. We therefore train a lightweight router $r_\phi(d \mid x)$ on the same domain-labelled data used to train the memories. Given an input x, the router selects

$$\hat{d} = \arg\max_{d\in\mathcal{D}} r_\phi(d \mid x),$$

and generation proceeds with $f_{\theta,z,\mathcal{M}_{\hat{d}}}$. The router only selects which memory to attach; it does not change the shared mask or the compact backbone. We describe it further in Appendix A.1.

3.2 Domain-specific memory modules

Each module $\mathcal{M}_d$ adds domain-specific capacity to the shared sparse backbone through two complementary mechanisms: a LoRA adapter and a learnable KV cache.

LoRA adapters. For each domain (or language) $d \in \mathcal{D}$, we introduce a LoRA adapter (Hu et al., 2022). The adapter provides a lightweight parameter-space correction to the pruned backbone. In our experiments, LoRA adapters target selected projection matrices and may be placed either in all transformer blocks or only in a contiguous subset of layers, such as the top N blocks. We discuss the choice in Section 4.4.

Learnable KV cache. Alongside the LoRA adapter, each domain d is equipped with a set of N_{kv} learnable key-value memory representations at each attention layer where the module is active. As in Eyuboglu et al. (2025), these keys and values are not produced by Transformer computation; they are trainable parameters directly inserted into the attention mechanism. Specifically, at layer l the memory consists of $K_d^{(l)}, V_d^{(l)} \in \mathbb{R}^{h \times N_{kv} \times d_h}$, where h is the number of KV heads and d_h is the head dimension. The memory keys and values are prepended to the regular text keys and values:

$$\tilde{K}^{(l)} = [K_d^{(l)};\ K_{\text{text}}^{(l)}],\ \tilde{V}^{(l)} = [V_d^{(l)};\ V_{\text{text}}^{(l)}]. \quad (3)$$

The causal mask is extended so that every text position can attend to all N_{kv} memory tokens, while the standard causal constraint is preserved among text positions. The memory parameters are initialized from $\mathcal{N}(0, 0.02)$ and trained jointly with the LoRA weights.

3.3 Learning a sparse backbone

We learn the shared mask using the L_0 regularization framework of Louizos et al. (2018). Each removable channel has a stochastic binary mask variable $z_i \in \{0, 1\}$, relaxed during training with the Hard Concrete distribution. The mask parameters define the distribution over retained channels, and are not input-dependent gates. This lets us optimize a structured sparsity pattern jointly with the domain modules, then fix and materialize it after training.

For a linear map with output dimension n_{out}, we learn $z \in \{0, 1\}^{n_{\text{out}}}$ over output channels. Let $\boldsymbol{\alpha}$ denote the Hard Concrete parameters. We penalize the expected number of retained channels,

$$\mathcal{R}(\boldsymbol{\alpha}) = \mathbb{E}_{\mathbf{z}}\big[\|\mathbf{z}\|_0\big],$$

and impose a target sparsity s^* through a hinge penalty on the active fraction:

$$\Omega(z) = \lambda\big[\hat{a} - (1 - s^*)\big]_+, \qquad \hat{a} = \frac{\mathcal{R}(\boldsymbol{\alpha})}{|\boldsymbol{\alpha}|}.$$

Here $[\cdot]_+ = \max(0, \cdot)$. The loss encourages the shared backbone to keep at most the target fraction of active channels, while the domain modules learn to recover capacity that can be externalized.

LoRA adapters modify the same linear maps that are being pruned, thus, the mask must define which output channels remain available to both the pretrained weights and the domain-specific correction. For a base linear map W and a domain-specific LoRA update ΔW_d, we therefore apply the mask after the two branches are added:

$$y = z \odot (Wx + \Delta W_d x).$$

This prevents the LoRA branch from reintroducing channels that have been removed from the shared backbone, and makes the final structure explicit: after training, channels with $z_i = 0$ can be removed from both the base map and the corresponding LoRA output dimensions.

3.4 Optimization protocol

Section 3.1 defines the objective; we now describe the training procedure. Throughout training, the pretrained parameters θ are frozen. We optimize only the shared mask and the domain-module parameters. This isolates the question we want to study: whether existing capacity can be relocated from the backbone into memory modules, using moderate amounts of domain data, while retaining the behaviour of the original model.

Both domain and retention batches use the same token-level distillation loss from the original unpruned model to the student, equivalently $\mathrm{KL}(p_\theta(\cdot|x) \,\|\, p_{\theta,z,\mathcal{M}_d}(\cdot|x))$. For each input, we store the teacher's top-100 logits and train the offloaded model to match the corresponding distribution. The balance between domain learning and retention is controlled by mixing domain and retention batches, rather than by introducing separate loss forms.

There is a slight difference in how domain and retention batches are processed in training. For a domain batch (x, d), we attach the corresponding

module $\mathcal{M}_d$ and update the shared mask together with that module, including the sparsity penalty. For a retention batch, x is drawn from general-domain data and has no domain label; we attach a randomly sampled module $\mathcal{M}_{\tilde{d}}$ and update only the shared mask. Thus, domain batches teach the modules to recover domain-specific capacity, while retention batches discourage the mask from removing broadly useful behaviour.[1]

4 Experiments

In our experiments, we ask whether domain-specific capacity can be moved from a frozen backbone into external memory modules while preserving broad model capabilities. This is not primarily a parameter-count reduction setting: if all modules are counted, the total parameter budget remains roughly comparable to the original model. The benefit is modularity and control. At inference time, only one memory module needs to be attached, and different users or deployments can access different capabilities while sharing the same core model.

We first describe the experimental setup, then evaluate topic-domain offloading (Section 4.2) and language offloading (Section 4.3). We then present ablations (Section 4.4) and specialization analyses of the learned backbone and modules (Section 4.5).

4.1 Experimental setting

Models We experiment with 3 pretrained base models spanning two model families and two scales: Llama-3.2-3B, Llama-3.1-8B (Touvron et al., 2023), and Qwen-2.5-3B (Yang et al., 2024). Base models provide a clean first setting for studying capacity allocation, since retention can be defined through language-model behavior; extending this to instruction-tuned or reasoning models may require preserving richer task behavior, reasoning traces, tool use, and safety-relevant responses.

Topic domains Our primary experiments use Wikipedia-Topics (Ziadé, 2024), which partitions English Wikipedia into 6 topic domains: Mathematics, Science, Technology, History, Culture, and Geography. We sample 30,000 articles per domain and apply a 95/5 train/evaluation split (approximately 150M tokens).

Language domains As a second partitioning strategy we use Wikisource (Wikimedia Foundation), selecting six languages with sufficient coverage: English, Spanish, French, German, Italian, and Portuguese. Each language is treated as a separate domain with its own memory module, and total volume of the training data is again roughly 150M tokens.

Retention corpora The retention loss (Section 3.1, $\mathcal{L}_{\text{retain}}$) draws mini-batches from a mixture of C4 (Raffel et al., 2020) (general English text), GSM8K (Cobbe et al., 2021) (grade-school math) and ARC-Challenge (Clark et al., 2018) (science reasoning). Such data should differ from Wikipedia articles and encourage retention of various capabilities (e.g., reasoning). During training, the models see roughly 9M tokens from these datasets

Setting and hyperparameters We prune the attention key/value projections (W_K, W_V) together with the MLP projections (W_{gate}, W_{up}, W_{down}); this is also where LoRA adapters are applied. We found that attaching the memory modules to higher layers is beneficial: for all experiments, we inject memory in the last blocks. We use a fixed LoRA rank of $r = 256$ and $N_{kv} = 256$ KV memory tokens per layer. This yields around 180M parameters per memory module for 3B models and 277M for 8B; in every case this is below the number of parameters pruned from the backbone at 12%. As only one memory module is used at inference, the effective parameter count remains roughly comparable to the unpruned baseline. See Appendix E for the full hyperparameter breakdown.

Evaluation We use three metrics: (i) per-domain held-out perplexity; (ii) MMLU (Hendrycks et al., 2021) (5-shot) for broad factual knowledge; and (iii) BigBench-Lite (BBL) (Suzgun et al., 2022) for multi-step reasoning.

4.2 Experiments with topic domains

Table 1 reports the main topic-domain results (left). We prune 12% of global parameters, applying the mask to the attention `k,v` projections and the MLP `gate,up,down` projections.[2] Since pruning is restricted to these parameter groups, the local pruning rate within them is substantially higher.

Across model families and scales, knowledge offloading preserves most of the unpruned model's

[1] The release including code, configuration files, and scripts for reproducing the experiments reported is available here.

[2] Note that while for unstructured pruning results are typically reported at 50% sparsity, structured pruning is considerably harder and studied at lower sparsity levels.

Model	Topic domains			Language domains		
	MMLU↑	BBL↑	PPL↓	MMLU↑	BBL↑	PPL_{avg} ↓
Llama-3B (unpruned)	0.560	0.393	8.74	0.560	0.393	15.40
Llama-3B (pruning only)	0.249	0.372	24.01	0.316	0.382	15.12
Llama-3B (KOFF)	0.520	0.470	8.85	0.484	0.412	14.95
Qwen-3B (unpruned)	0.644	0.410	14.14	0.644	0.410	13.42
Qwen-3B (pruning only)	0.329	0.360	32.41	0.326	0.374	12.10
Qwen-3B (KOFF)	0.610	0.668	13.75	0.413	0.380	10.38
Llama-8B (unpruned)	0.667	0.435	7.93	0.667	0.492	9.62
Llama-8B (pruning only)	0.408	0.353	19.78	0.384	0.364	9.54
Llama-8B (KOFF)	0.603	0.480	6.84	0.415	0.386	9.03

Table 1: Topic-domain and language-domain results, at global sparsity level 12%. Pruned variants use the same L_0 sparsification objective but do not have memory modules. PPL_{avg} is perplexity averaged over 6 considered languages.

MMLU performance and keeps perplexity close to the dense baseline. In contrast, the pruning-only baselines, which learn the same L_0 mask but use no memory modules, degrade sharply, especially on MMLU and perplexity. Thus, the gains do not come from structural pruning alone: at this sparsity level, pruning a frozen model without a recovery mechanism is highly damaging. The memory modules are essential for restoring capacity removed from the shared backbone.

The BBL results are particularly interesting. Knowledge offloading improves BBL over the unpruned model in all three cases, even though the training objective is teacher distillation from the original model and is mainly designed to preserve behavior, not improve it. This suggests that domain modules, apart from repairing pruning damage, can also provide useful domain-conditioned adaptation.

In Figure 2 we study the sparsity–quality trade-off. The curve shows a clear operating regime: performance degrades gradually up to around 12% global sparsity, but drops sharply afterwards. At 5–12% sparsity, MMLU remains close to the unpruned baseline, perplexity changes only mildly, and BBL outperforms the dense model. By contrast, at 20% sparsity, MMLU and perplexity deteriorate substantially, and the memory modules no longer compensate for the removed capacity. Thus, we use 12% global sparsity as target high enough to test non-trivial capacity relocation, but below the sharp degradation regime.

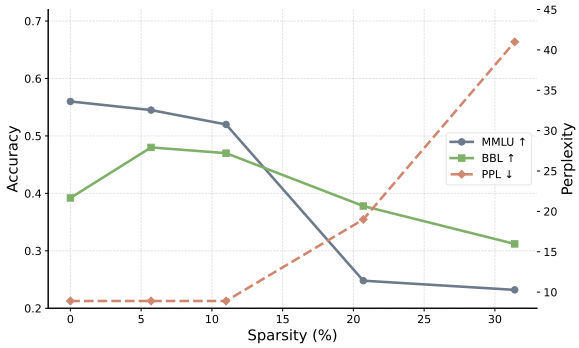

Figure 2: Sparsity–quality trade-off for Llama-3.2-3B, with topic domains.

4.3 Multilingual experiments

As a complementary setting, we also evaluate KOFF on language domains with the same set of base models (Table 1, right). We use the same global sparsity level as in the topic-domain experiments, approximately 12%, but define domains by language rather than subject: English, Spanish, French, German, Italian, and Portuguese. This setting introduces a stronger distribution shift, since the modules must account not only for different content but also for different token distributions.

We treat held-out perplexity as the main evaluation signal, averaged across the six languages and computed with the matched language module. MMLU and BBL are included only as English-side checks of retained broad capability. Table 1 shows that the same sparse-backbone-plus-memory decomposition remains viable under this language-based partition. We use this setting primarily as a

stress test beyond topical domains; in Section 4.5.2, we further analyze whether the learned mask indeed targets language-specific capacity.

4.4 Ablations

Table 2 ablates the main components of KOFF, in the topic-domain setting, on Llama-3.2-3B at 12% global sparsity. The pruning-only baseline confirms that sparsifying a frozen model without a recovery mechanism is highly damaging. Both recovery mechanisms help, and their effects are complementary: LoRA is more effective than the KV cache in this setting, but combining the two gives the best overall result.

We also compare against a two-stage sequential version of KOFF ('prune → recover'): first prune the model, then freeze the pruning mask and train the memory modules afterwards. This is slightly weaker than joint training. In this setting, the mask is learned without knowing which removed capacity can later be recovered by memory, so it may discard information that is hard to reconstruct. Joint training avoids this by letting pruning and recovery co-adapt.[3]

The placement of the memory modules also matters. Attaching them only to the upper layers outperforms the all-layer variant. This suggests that memory is most useful when it refines higher-level representations rather than intervening throughout the network. This points to a natural extension of our framework: instead of manually choosing the target layers or fixing sparsity pattern, one could learn where capacity should remain in the backbone and where it should be offloaded.

Finally, the retention objective has little effect in this configuration: removing it gives nearly identical numbers. This does make retention unnecessary in general. Appendix D shows mild but consistent MMLU gains in a broader grid search, suggesting that the effect is dependent on pruning regime and the choice of retention data. Defining retention objectives that better capture the capabilities one wants to preserve remains an open design question.

4.5 Specialization analysis

We now look beyond aggregate performance and ask whether the learned decomposition produces a separation of roles: which information remains in the shared backbone, and which is carried by the domain-specific memory modules.

[3]A dense LoRA+KV distillation baseline without pruning is not included. As the unpruned backbone is the KL teacher, modules that leave the model unchanged receive no useful gradient, while perturbing modules (e.g., randomly initialized) are trained to undo the perturbation, collapsing to 'Unpruned'.

Configuration	L0	LoRA	KV	MMLU↑	BBL↑	PPL↓
Unpruned	✗	✗	✗	0.56	0.39	8.7
Pruning only	✓	✗	✗	0.25	0.37	24.0
Pruning + LoRA	✓	✓	✗	0.49	0.45	9.0
Pruning + KV cache	✓	✗	✓	0.41	0.44	17.5
KOFF (LoRA + KV)	✓	✓	✓	**0.52**	**0.47**	**8.8**
KOFF: prune → recover	✓	✓	✓	0.50	0.41	8.9
KOFF: all layers	✓	✓	✓	0.47	0.44	9.1
KOFF: no retention	✓	✓	✓	0.52	0.47	8.8

Table 2: Ablations with Llama-3.2-3B at 12% sparsity, with topic domains.

4.5.1 Modules specialize to domains

We first test whether memory modules are functionally domain-specific. In the language-domain setting, we evaluate each language with every memory module attached to the shared backbone. If modules store domain-specific capacity, the matched module should give the best perplexity, while mismatched modules should degrade performance.

Figure 3 shows this pattern: perplexity is lowest on the diagonal, where the evaluation language and memory module match, and increases when the wrong module is attached. This suggests that the modules are not interchangeable adapters; they recover information specific to their own domains. Interestingly, the off-diagonal structure does not show a clear typological pattern: related languages do not consistently benefit from each other's modules. For example, Romance-language modules are not uniformly better for other Romance languages, nor do English and German form a distinct cluster.

4.5.2 Language-specific capacity is preferentially offloaded

Having shown that the modules specialize functionally, we now ask whether the pruning mask shows a corresponding structural specialization. We compare the MLP channels removed by our pruning procedure with the neuron ranking induced by LAPE (Tang et al., 2024), which orders neurons from more language-specific to language-general. We treat the top 15% most language-specific neurons as language-specific, and the top 15% most language-general neurons as language-general.

Table 3 shows a clear separation. Language-specific neurons are removed more often than the average pruning rate: around 67–71% are pruned. In contrast, language-general neurons are strongly

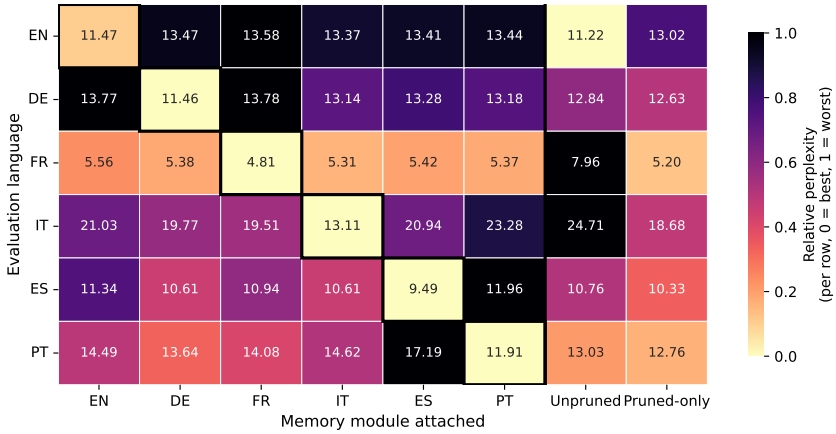

Figure 3: Cross-language memory-module swap on Llama-3.2-3B Each cell reports held-out perplexity on the row's evaluation language when the column's memory module is attached to the shared backbone.

Pruned in	Sparsity s	P(pruned|spec.)	P(pruned|gen.)	P(gen.)/s
`gate_proj`	0.50	0.67	0.04	0.09
`up_proj`	0.52	0.69	0.05	0.09
either	0.53	0.71	0.05	0.09

Table 3: LAPE cross-reference for a language-domain run on Llama-3.2-3B. Pruning primarily removes language-specific MLP channels while largely preserving language-general ones.

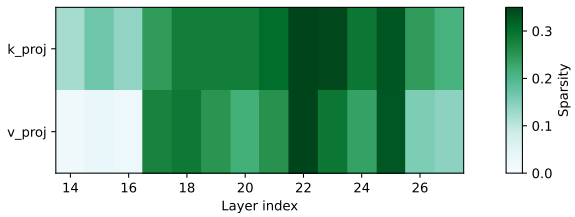

Figure 4: Per-layer sparsity for `k_proj` and `v_proj` layers of Llama-3.2-3B.

protected: only 4–5% are removed, even though the relevant projections are pruned at roughly 50% local sparsity. Thus, instead of pruning uniformly, KOFF preferentially removes language-specific capacity while preserving the cross-lingual core. This is consistent with memory modules recovering offloaded language-specific information.

4.5.3 What remains in the backbone

Finally, we look at *which* parts of the network the learned mask removes from the shared backbone. Figure 4 shows the per-layer final sparsity of `k_proj` and `v_proj` for Llama-3.2-3B. Two patterns stand out. First, the earlier and later layers are pruned least, and most of the pruning happens in middle layers. Second, `k_proj` is pruned more aggressively than `v_proj` across most layers.

As a complementary check, we verify that the backbone remains close to the original model in representation space: linear CKA (Kornblith et al., 2019), kNN overlap, and class-centroid cosine similarity all remain close to 1 at 12% global sparsity. Details are given in Appendix B.

5 Conclusion

We introduced a knowledge offloading framework (KOFF): a way to split a pretrained language model into a shared sparse backbone and small, swappable memory modules. The base model parameters remain fixed; we only learn which capacity can be removed from the core and recovered through external memory, implemented here with LoRA adapters and learned KV caches.

The main result is that this can be done without a large loss in model ability. Across topic domains, knowledge offloading preserves much of the unpruned model's MMLU performance, keeps perplexity close to the dense baseline, and improves BBL. Pruning the same frozen model without memory modules causes much larger degradation, suggesting that some domain-specific knowledge can be moved out of the shared backbone and restored when the corresponding module is attached.

This gives a more modular view of model capacity. Rather than treating all knowledge as inseparably stored in one parameter set, we can keep a broadly useful core and attach domain-specific memories as needed. Our analyses support this picture: matched modules work best, language-specific neurons are preferentially removed while language-general neurons are protected, and the backbone remains close to the original representation geometry.

6 Limitations

We study a limited set of domains and moderate-size models. However, different types of knowledge (e.g., factual, linguistic, stylistic, procedural, or safety-relevant) may differ in how easily they can be moved into memory. Moreover, we assume that domains are predefined: each memory module corresponds to a given topic or language label. A natural extension would be to discover this structure automatically, learning both the domain partition and the allocation of capacity between the shared backbone and the memory modules. We leave scaling to larger models, broader domain partitions, and more heterogeneous knowledge for future work.

We also focus on pretrained base models only. This provides a clean initial setting as retention can be defined through language-model behavior. For instruction-tuned, reasoning, or agentic models, retention may need to preserve richer behavior: instruction following, reasoning traces, tool use, refusal behavior, or other safety-relevant responses.

In addition, our analysis provides evidence of specialization, but a deeper mechanistic study could investigate which knowledge is retained in the backbone, which moves into memory, and how this depends on layer, sparsity, module type, and objective.

Finally, one natural use case of KOFF is controlled access to capabilities: a sensitive domain module could be available only to trusted users, while the shared backbone remains broadly deployable. Testing this setting would require adversarial evaluation, checking not only ordinary use but also prompting attacks, distribution shift, module substitution, and interactions with tools or other memories. We leave this for future work.

References

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. 2024. Slicegpt: Compress large language models by deleting rows and columns. *Preprint*, arXiv:2401.15024.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, and 1 others. 2025. Cartridges: Lightweight and general-purpose long context representations via self-study. *arXiv preprint arXiv:2506.06266*.

Negar Foroutan, Mohammadreza Banaei, Rémi Lebret, Antoine Bosselut, and Karl Aberer. 2022. Discovering language-neutral sub-networks in multilingual language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 7560–7575.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Preprint*, arXiv:2009.03300.

Amr Hendy, Mohamed Abdelghaffar, Mohamed Afify, and Ahmed Y Tawfik. 2022. Domain specific sub-network for multi-domain neural machine translation. In *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 351–356.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *Preprint*, arXiv:2106.09685.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations (ICLR)*.

Hanyu Hu, Pengxiang Zhao, Ping Li, Yi Zheng, Zhefeng Wang, and Xiaoming Yuan. 2025. Fasp: Fast and accurate structured pruning of large language models. *Preprint*, arXiv:2501.09412.

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. 2023. Sparse upcycling: Training mixture-of-experts from dense checkpoints. In *International Conference on Learning Representations*.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. 2019. Similarity of neural network representations revisited. In *International conference on machine learning*, pages 3519–3529. PMLR.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3045–3059.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33.

Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A. Smith, and Luke Zettlemoyer. 2022. Branch-train-merge: Embarrassingly parallel training of expert language models. *Preprint*, arXiv:2208.03306.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *Preprint*, arXiv:2101.00190.

Christos Louizos, Max Welling, and Diederik P. Kingma. 2018. Learning sparse neural networks through l_0 regularization. *Preprint*, arXiv:1712.01312.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Preprint*, arXiv:2305.11627.

Hadi Pouransari, David Grangier, C Thomas, Michael Kirchhof, and Oncel Tuzel. 2025. Pretraining with hierarchical memories: separating long-tail and common knowledge. *Preprint*, arXiv:2510.02375.

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

Sainbayar Sukhbaatar, Olga Golovneva, Vasu Sharma, Hu Xu, Xi Victoria Lin, Baptiste Rozière, Jacob Kahn, Shang-Wen Li, Wen-tau Yih, Jason E. Weston, and Xian Li. 2024. Branch-train-mix: Mixing expert llms into a mixture-of-experts llm. In *First Conference on Language Modeling*.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. End-to-end memory networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 28.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *Preprint*, arXiv:2210.09261.

Tianyi Tang, Wenyang Luo, Haoyang Huang, Dongdong Zhang, Xiaolei Wang, Wayne Xin Zhao, Furu Wei, and Ji-Rong Wen. 2024. Language-specific neurons: The key to multilingual capabilities in large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5701–5715.

Hugo Touvron, 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. *Preprint*, arXiv:2302.13971.

Wikimedia Foundation. Wikisource: The free library. `https://wikisource.org`. Multilingual collection of source texts.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 43 others. 2024. Qwen2 technical report. *Preprint*, arXiv:2407.10671.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2022. MoEfication: Transformer feed-forward layers are mixtures of experts. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 877–890, Dublin, Ireland. Association for Computational Linguistics.

Tarek Ziadé. 2024. Wikipedia-topics dataset. Hugging Face Datasets. Wikipedia articles partitioned into 40 root categories.

A Training hyperparameters

Table 4 lists the default configuration used in all topic-domain experiments unless stated otherwise. All runs use the hinge ℓ_0 loss (Eq. 1), outer gate placement, and a target per-module sparsity of 15%. Three separate AdamW optimizers handle the LoRA weights ($\eta = 10^{-5}$), gate parameters ($\eta = 10^{-2}$), and memory bank ($\eta = 10^{-2}$). For the 8B models we lower λ to 25 and the retention interval to 20 steps to account for the larger gate count and batch size of 1. All models are loaded in bfloat16; the 8B variants use gradient checkpointing to fit within 80 GB GPU memory. We tokenize texts with each model's native tokenizer (maximum sequence length 512).

Parameter	Value
LoRA rank r / scaling α	256 / 128
LoRA target modules	K, V, Gate, Down, Up
LoRA learning rate η_{LoRA}	1×10^{-5}
Gate learning rate η_{gate}	1×10^{-2}
Memory KV cache learning rate η_{mem}	1×10^{-2}
L0 λ (with auto-scaling)	60
Target sparsity s^* (local, per-module)	0.15
Temperature β	2.0 (fixed)
Memory tokens M	256 per domain
Batch size	2
Max sequence length	512
retention-loop interval S	every 20 steps
Epochs	1

Table 4: Default hyperparameters (3B, topic-domain setting)

A.1 Domain classification model for inference routing

We train a RoBERTa on the same dataset as the main model (Wikipedia-Topics for topic-domain runs, WikiSource for language-domain runs) to classify texts into domains. All parameters are unfrozen, the learning rate is set to 0.0003, and the batch size to 64.

B Backbone probe protocol

Backbone probes are run with adapters disabled and no memory module injection, so all measurements target only the shared pruned backbone. We use the same six-topic Wikipedia-Topics partition as in the main experiments (Mathematics, Science, Technology, History, Culture, Geography), sample up to 5000 texts per topic, and split them 80/20.

For representation similarity we report linear CKA (Kornblith et al., 2019) between base and pruned test embeddings, along with two neighborhood/cluster diagnostics (kNN-overlap and class-centroid cosine similarity). These diagnostics are intended to check whether pruning preserves global embedding geometry.

C Experiments on 1B models

Tables 5–8 report experiments on Llama-3.2-1B that guided early design decisions (L0 loss variant, pruning target selection, multilingual scaling, data volume). All 3B and 8B experiments in the main text use the hinge loss motivated by these preliminary runs, gating the `k`, `v`+MLP projections.

L0 Variant	λ	Rank	Final Sparsity	PPL (en)↓
Hinge	60	128	0.197	32.9
Hinge	60	64	0.198	33.6
Hinge	15	128	0.150	17.8
Hinge	10	64	0.073	28.8
Progressive	15	128	0.105	19.5
Progressive (ReLU)	15	128	0.120	19.3
Lagrangian	60	128	0.984	340.4
Lagrangian	10	64	0.986	365.9

Table 5: L0 loss variant comparison on Llama-3.2-1B (English, target local sparsity 0.2)

Target Modules	Final Sparsity	PPL (en)↓	Steps
MLP (gate, up, down)	0.150	17.8	9644
KV (k_proj, v_proj)	0.202	17.6	1280

Table 6: Pruning convergence speed by module type on Llama-3.2-1B (hinge loss, rank 128, target local sparsity 0.2). Final sparsity is reported locally (within the gated modules).

D Retention loop ablation

Retention loop ($\mathcal{L}_{\text{retain}}$). Structured pruning inevitably removes capacity from the base model. To prevent the gates from over-pruning neurons that are critical for general-purpose reasoning, we interleave the main training loop with periodic retention steps that realize the $\mathcal{L}_{\text{retain}}$ term in Eq. 1. Every S main training steps the following procedure is executed:

1. A random LoRA and KV-cache pair is inserted into a model
2. A fresh mini-batch is drawn from a mixture of general-domain corpora: C4 (English, streamed), GSM8K (math reasoning), and ARC-Challenge (science reasoning).

Languages	s^*	Final Sparsity	PPL (en)↓	PPL (fr)↓	PPL (es)↓	PPL (avg)↓
en	0.2	0.150	17.8	–	–	17.8
en, fr	0.1	0.071	19.7	9.8	–	14.8
en, fr, es	0.1	0.073	18.5	11.2	20.9	16.8
en, fr, es	0.2	0.146	20.3	11.1	22.1	17.8

Table 7: Multilingual pruning on Llama-3.2-1B with shared L0 gates and per-language LoRA adapters (hinge loss, rank 128, MLP modules). Target s^* and final sparsity are reported locally (within the gated MLP projections).

Records	Epochs	Final Sparsity	PPL (en)↓
5K	3	0.196	51.3
10K	5	0.150	17.8
10K	10	0.146	20.5
20K	5	0.149	12.3

Table 8: Effect of training data scale on Llama-3.2-1B (English, hinge loss, rank 128, target local sparsity 0.2). Final sparsity is reported locally.

3. A standard language-modeling forward–backward pass is performed; gradients flow to the gate parameters and memory modules.

This strategy exposes the gates to data that is distribution-shifted relative to the domain-specific training data, providing a regularizing signal that discourages pruning neurons whose removal would hurt broad language understanding.

Table 9 gives the retention-loop comparison under temperature annealing (β: $2.0 \rightarrow 0.5$), crossing module set (k,v vs. k,v+MLP) with regularisation strength ($\lambda \in \{60, 120\}$).

Two patterns are worth highlighting. First, enabling the retention loop generally improves MMLU across all four settings: for k,v, MMLU rises from 0.520 to 0.540 at λ=60 and from 0.560 to 0.568 at λ=120; for k,v+MLP, it rises from 0.504 to 0.520 at λ=60 and from 0.503 to 0.514 at λ=120. Second, BBL changes are smaller and mixed at λ=60 ($0.480 \rightarrow 0.482$ for k,v, $0.472 \rightarrow 0.470$ for k,v+MLP), while at λ=120 both module sets improve slightly ($0.510 \rightarrow 0.530$ for k,v, $0.468 \rightarrow 0.470$ for k,v+MLP). Perplexity is consistently higher with retention in this grid (e.g., $8.69 \rightarrow 8.85$ and $8.78 \rightarrow 8.90$), suggesting a mild trade-off toward benchmark retention.

E Distillation details

Table 10 collects all distillation results across models and domain types in one place. For all experiments, we use top-100 logit KL. The blended variant (α=0.5, mixing cross-entropy and KL) diverges in every case. Teacher-only KL, by contrast, is the strongest objective for preserving MMLU: in the language-domain setting it prevents the English adapter from losing general knowledge to the multilingual training signal, and in the topic-domain setting, combined with hyperparameter search, it improves MMLU over plain SFT (0.520 vs. 0.482 on Llama-3.2-3B).

We suspect the blended loss fails because the KL term and the cross-entropy term pull the adapter in opposite directions when the pruned model's output distribution has already shifted substantially from the teacher's due to KV-cache blocks. At 20% sparsity the logit landscape is different enough that the KL gradient dominates early training, preventing the adapter from learning useful domain-specific corrections.

F Generalization across model family and size

Here we extend the generalization result presented in §4.5.1 to two more axes: model family (§F.1) and scale (§F.2)

F.1 Across Model Families

The Llama-3.2-3B and Qwen-2.5-3B rows in Table 1 show that the same recipe transfers across families. The Qwen result, where the pruned model far exceeds the unpruned baseline on BBL (0.668 vs. 0.410) while nearly matching it on MMLU (0.610 vs. 0.644), is consistent with the LoRA-only baseline on Qwen showing the same pattern: most of the gain is format adaptation rather than pruning, and pruning + memory modules does not undo it. The variance across three seeds ($\pm$0.005 MMLU, $\pm$0.013 BBL) is small enough that the result is reproducible.

F.2 Across Model Scales

Llama-3.1-8B at $\sim$12% global sparsity (with teacher-only KL distillation, our default) reaches MMLU = 0.603 and PPL = 6.84, the lowest perplexity in our experiments. The backbone alone (all adapters and memory banks disabled) retains

Modules	λ	Retention	MMLU↑	BBL↑	PPL↓	Sparsity
k,v	60	Off	0.520	0.480	8.69	0.022
		On	0.540	0.482	8.85	0.022
	120	Off	0.560	0.510	8.91	0.022
		On	0.568	0.530	8.97	0.022
k,v+MLP	60	Off	0.504	0.472	8.80	0.122
		On	0.520	0.470	8.85	0.120
	120	Off	0.503	0.468	8.78	0.120
		On	0.514	0.470	8.90	0.121

Table 9: Retention-loop ablation with sparsity aligned to the main results/module-anatomy tables.

Model	Domain	Loss	MMLU↑	BBL↑	PPL↓	Sparsity
Llama-3.2-3B	Topic	SFT	0.482	0.446	8.77	0.121
	Topic	Blended (α=0.5)	0.304	0.350	86.42	0.125
	Topic	Teacher-only KL	0.520	0.470	8.85	0.120
Qwen-2.5-3B	Topic	SFT	0.564	0.620	13.73	0.121
Qwen-2.5-3B	Topic	Blended (α=0.5)	0.310	0.345	54.7	0.124
Qwen-2.5-3B	Topic	Teacher-only KL	0.610	0.668	13.75	0.122
Llama-3.1-8B	Topic	SFT	0.574	0.441	6.70	0.121
Llama-3.1-8B	Topic	Blended (α=0.5)	0.340	0.420	35.12	0.120
Llama-3.1-8B	Topic	Teacher-only KL	0.603	0.480	6.84	0.121

Table 10: Distillation results across all settings. Blended training diverges universally. Teacher-only KL preserves/improves MMLU in both language- and topic-domain settings.

Model	Distill	MMLU↑	BBL↑	PPL↓	Sparsity
Llama-3.1-8B	Teacher-KL (ours)	**0.603**	**0.480**	**6.84**	0.121
Llama-3.1-8B	Blended (α=0.5)	0.340	0.420	35.12	0.120

Table 11: 8B model results (topic-domain, k,v+MLP modules, rank 256, ∼12% global sparsity). The default uses teacher-only KL distillation; blended distillation diverges.

MMLU = 0.538, suggesting that at 8B scale most of the pruning damage is absorbed by the base model's redundancy rather than by the recovery modules. This is consistent with the backbone-similarity check in Section 4.5.3: backbone geometry is preserved at 3B scale already, and at 8B scale the backbone alone is a competent model. Per-topic perplexities are low and uniform.

Blended distillation fails here too: α=0.5 diverges, dropping MMLU to 0.340 and raising perplexity to 35.12 at the same ∼12% global sparsity. We used reduced regularisation (λ=25, retention interval 20, batch size 1) to accommodate the larger gate count (Table 11).

G Memory module placement in the model

G.1 Two placement decisions.

Capacity relocation introduces two placement decisions that are absent in conventional single-network pruning: (a) *which weights* the shared mask $\mathbf{z}$ acts on (e.g., attention projections vs. MLP projections), and (b) *which layers* the memory module parameters $\theta^{(d)}_{\text{cart}}$ are attached to (all transformer blocks, or a subset). Both choices interact with the recovery budget and are studied empirically in Section G.2. The specific instantiation below (LoRA + KV memory bank as the memory module; Hard Concrete neuron gates as the mask) is one practical realization of this framework.

G.2 Which Modules to Prune

The first design decision is which weight matrices to gate, and it is constrained by where the prunable parameters live. On Llama-3.2-3B the MLP projections (gate, up, down) hold roughly three quarters of the non-embedding weights, so a mask that gates only the attention projections cannot remove enough of the model to reach a meaningful *global* sparsity. Gating k,v alone saturates at 2.2% global sparsity and q,k,v at 2.8%, far below our

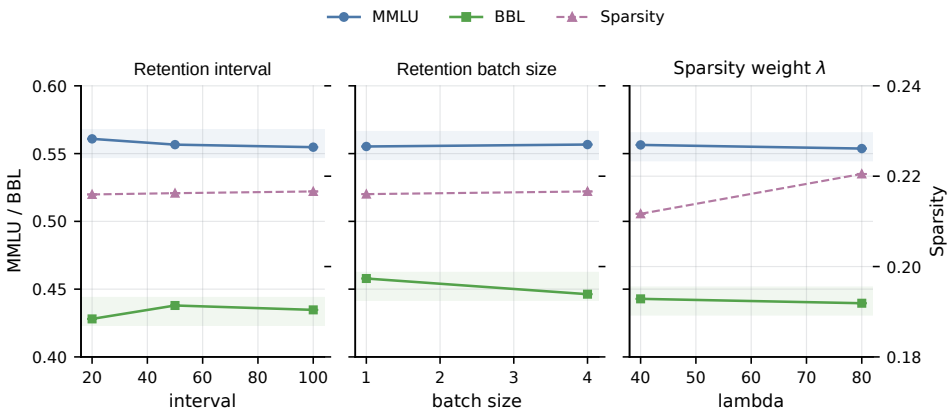

Figure 5: Single-factor slices for the upper-layer sweep: retention interval (A), retention batch size (B), and sparsity weight λ (C). MMLU and BBL share the primary axis, sparsity is plotted on a secondary axis, and shaded bands indicate the local noise scale around each series mean. Across all three factors, shifts are modest and support the conclusion that these hyperparameter effects are second-order relative to layer placement.

Target Modules	MMLU↑	PPL↓	Sparsity
k,v+MLP	**0.520**	**8.85**	**0.120**
k,v only	0.488	10.26	0.022
q,k,v	0.490	10.16	0.028

Table 12: Module-set comparison on Llama-3.2-3B (topic-domain, rank 256, ∼12% global-sparsity target). Pruning the MLP is required to reach a meaningful global sparsity: attention-only gating saturates far below target and yields lower MMLU at higher perplexity.

∼12% target; forcing those small budgets harder is destructive, so even at saturation they trail the KV+MLP configuration by 3.2 and 3.0 points of MMLU and 1.41 and 1.31 of perplexity (Table 12). Gating `k,v` together with the MLP projections spreads the cut across many more neurons, reaching 12% global sparsity at higher MMLU (0.520) and lower perplexity (8.85). We leave `o_proj` out: because it writes directly into the residual stream, gating its outputs corrupts the shared representation and collapses quality. As a single-run sanity check on Llama-3.2-3B, adding `o_proj` to the gated set under our default configuration drops MMLU to 0.23 (near-random), against 0.520 for the matched `k,v`+MLP setup of Table 12; we therefore exclude it from all reported configurations. We likewise leave `q_proj` out, as it adds little prunable budget and its benefit is at best marginal and architecture-dependent (on Qwen-2.5-3B the QKV gap nearly closes). MLP gates converge more slowly than attention gates—about 9,644 versus 1,280 steps to target (Table 6)—so we allow a longer schedule and stronger regularisation when the MLP is gated.

G.3 Which Layers to Attach Memory Modules to

A memory modules can in principle be attached at every transformer block, but doing so spreads its parameter budget thinly and lets domain-specific gradients reach early layers whose representations are useful to many domains. To study this, we vary the memory modules placement on Llama-3.2-3B between the upper N blocks of the 28-layer stack and all 28 blocks, holding the rest of the pipeline (`k,v`+MLP, rank 256, 256 memory tokens, ∼12% global sparsity) fixed. Restricting memory modules to the upper 18 blocks recovers MMLU = 0.520 and PPL = 8.85, against MMLU = 0.470 and PPL = 9.08 for the matched all-layer control: an MMLU gain of +0.050 and a lower perplexity. BBL also improves modestly (0.470 vs. 0.438). The interpretation lines up with the capacity-relocation framing: the lower layers carry features that many domains share, so freezing them and concentrating the parameter budget on the specialization-heavy upper half preserves more of the backbone's general reasoning, without sacrificing per-domain performance.

We then ran an experiment grid around the upper-layer configuration, varying the retention-loop interval (every 20, 50, or 100 main steps), the retention-loop batch size (1, 2, or 4), and λ (40, 60, or 80). Sparsity in this grid is reported as the *per-module (local)* target, fixed at 0.21 ± 0.01 across all thirteen runs. The trade-off grid is robust: every point has MMLU above 0.54 and BBL between 0.43 and 0.46. Single-factor slices in Fig. 5

show three patterns. Shorter retention intervals give slightly higher MMLU (0.561 at interval 20 vs. 0.555 at interval 100). Smaller retention batches give slightly higher BBL (0.458 at batch size 1 vs. 0.446 at batch size 4) at no cost to MMLU. And λ=40 slightly dominates λ=80 on both MMLU and BBL because it lands at a marginally lower local sparsity ($\sim$0.212 vs. $\sim$0.220). None of these is a phase transition; the upper-layer placement is the dominant design choice and the retention-loop hyperparameters are second-order.