KNOWLEDGE OFFLOADING: DECOMPOSING LLMS INTO SPARSE BACKBONES AND MEMORY MODULES

Anonymous authors
Paper under double-blind review

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 either jointly or sequentially 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 14B, we find that 12% to 20% of capacity can be moved out of the shared backbone while largely preserving perplexity and MMLU performance (up to 90% of the full model). We show that LoRA and learned KV memories are complementary, and analyses suggest that the offloaded knowledge is specialized: mismatched memories do not yield the same gains, and language-specific neurons are preferentially removed while language-neutral neurons largely remain in the backbone. These results suggest that the knowledge of a pretrained LLM can be reallocated between a shared core and swappable external memories.[1]

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. Yet, LLMs lack explicit mechanisms to manage how capacity is allocated across different types of knowledge. All information, both general and highly specialized, is stored within a single set of parameters, making it difficult to update, remove, or extend specialized knowledge without risking interference with unrelated capabilities. This motivates a modular perspective (Pfeiffer et al., 2023), in which knowledge is organized into separate, manageable components rather than a single undifferentiated set of parameters. Thus, we raise 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 that can be managed independently of the core?

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., 2026), and parameter-efficient adapters (Hu et al., 2022) 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 backbone 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

[1] Code for reproducing the experiments will be available here.

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

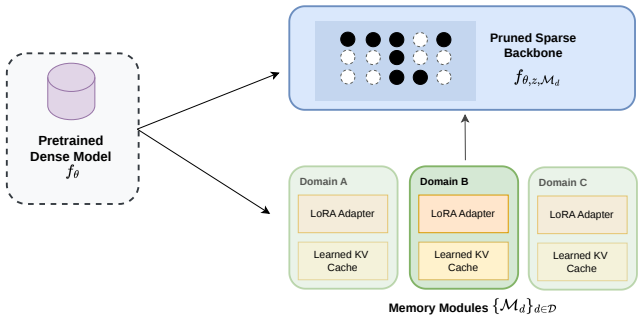

Figure 1: Knowledge offloading (KOFF) decomposes a pretrained LLM into a sparse backbone and separate swappable memory modules.

external memory. Instead of compressing a model into a single smaller network, KOFF decomposes a pretrained LLM into a shared sparse backbone and domain-specific memory modules: a structured pruning mechanism based on Hard Concrete gates (Louizos et al., 2018) determines which neurons are retained in the backbone, while memory modules are trained to compensate for the removed capacity through two recovery mechanisms: (1) a LoRA adapter (Hu et al., 2022) operating in parameter space, and (2) a learnable key-value (KV) cache injected into the attention mechanism (Eyuboglu et al., 2026), acting as an externalized knowledge store. We learn this decomposition either sequentially (KOFF: sequential), pruning first and then training the modules on the frozen backbone, or jointly (KOFF: joint), so that the mask can account for which capacity the modules can recover. Comparing the two lets us test whether deciding what to offload benefits from knowing what the modules can recover. At inference time, a small trained classifier selects the memory module to attach to the backbone for a given input.

We test KOFF on 4 LLMs ranging in size from 3B to 14B, using topic and language domains. Note that our goal is not to develop a more efficient pruning method instead, we use pruning without memory modules as a control. If specialized knowledge can be offloaded, attaching the matched memory module to a sparse backbone should recover more of the full model's performance than the backbone alone. Our results support this, at 20% sparsity, attaching the matched memory module consistently lowers perplexity relative to the pruning-only baseline and often improves MMLU, even when domains of different types are mixed. On Global MMLU, pruning alone already retains most of the full model's performance while perplexity rises substantially, consistent with the backbone retaining knowledge shared across languages. Neither variant consistently dominates: KOFF: joint is better on Llama-3.1-8B, while KOFF: sequential gives lower perplexity on Qwen models. For the instruct model, language modules trained on raw text reduce perplexity but hurt Global MMLU, showing that the modules offload whatever the domain data represents, including its format. Finally, ablations show that LoRA adapters and learned KV caches are complementary recovery mechanisms.

Importantly, recovering performance is a necessity but does not show that recovered knowledge is actually domain-specific. Our specialization analyses indicate that the decomposition reflects the intended split between shared and specialized knowledge. Functionally, the modules are domain-specific: attaching a mismatched module yields worse perplexity than the matched one. Structurally, the pruning mask removes most language-specific neurons from the backbone, while language-neutral neurons are almost entirely retained. Together, these results suggest that the capacity of a pretrained LLM can be reorganized into a shared core and separately managed specialized modules. To our knowledge, this is the first work to treat pruning as a mechanism for learning memory allocation.

2 Related Work

Memory-augmented models. Prior work extends LLM capacity by externalizing knowledge in activation or parameter space: prefix and prompt tuning inject learnable representations into the model's computation (Li & Liang, 2021; Lester et al., 2021), while LoRA (Hu et al., 2022) adds lightweight parameter updates. Drawing inspiration from these works, we let the model learn how to offload knowledge to such external memory. LoRA adapters provide parameter-space memory, while KV caches act as activation-space memory. Moreover, in multilingual NLP, modularity has been achieved by adding language-specific adapters to a frozen model (Pfeiffer et al., 2020), or by identifying

language subnetworks through pruning (Foroutan et al., 2022; Choenni et al., 2024). KOFF combines both, removing language-specific capacity from the backbone and relocating it to external modules. Recent work also externalizes factual knowledge into editable databases during pretraining (Zhao et al., 2026), trains compressed KV caches offline via context distillation (Eyuboglu et al., 2026), and shows that sparse memory-layer updates enable continual learning without catastrophic (Lin et al., 2025). Closest to our work are parametric memory approaches such as memory networks (Sukhbaatar et al., 2015) and hierarchical parametric memory (Pouransari et al., 2026), which learn memory alongside the model or from scratch. We instead start from a pretrained dense model, prune it, and use memory modules to recover knowledge lost during pruning. Unlike prior work, we treat memory as the recovery mechanism, where pruning and memory can be learned in sequence or jointly.

Model pruning. Prior work has shown that LLMs can be compressed via post-training structured pruning. 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 popularized the prune-then-adapt paradigm, where a pruned model is subsequently recovered using parameter-efficient methods such as LoRA. However, these approaches produce a single model with a fixed quality–efficiency trade-off. We instead use structured pruning to reorganize 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. Our sequential variant can be read as a domain-wise generalization of prune-then-adapt, recovering the removed capacity into several swappable modules rather than into one model, while our joint variant additionally lets the pruning mask depend on what those modules can recover.

Conditional computation and expert models. Dense-to-MoE conversion and expert-LM methods: MoEfication (Zhang et al., 2022), upcycling (Komatsuzaki et al., 2023), Branch-Train-Merge (Li et al., 2022) and Branch-Train-MiX (Sukhbaatar et al., 2024) introduce conditional specialization by constructing or routing expert computation. Our pruning objective instead makes the allocation of capacity itself the object of learning, and our modules are memory components attached to a shared backbone rather than routed FFN experts.

3 METHODOLOGY

3.1 GENERAL FRAMEWORK

KOFF relies on a separation of roles: the shared backbone preserves computation that is broadly useful across domains, while domain modules provide capacity for domain-specialized information. We consider two ways of learning this separation. In KOFF: joint, the pruning mask is trained 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. In KOFF: sequential, the mask is first learned without modules and then frozen, after which the modules are trained on top of the fixed backbone. This is simpler and decouples the two stages, but the mask cannot anticipate what the modules will be able to recover. Each 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 § 3.2 and are both shown beneficial in § 5.4.

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 to 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, on the domain data. At a high level, the objective is

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

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 (§ 3.3). KOFF: joint optimizes Eq. 1 directly. KOFF: sequential splits it into two stages: the mask is first learned with no modules attached, minimizing the same objective with $f_{\theta,z}$ in place of $f_{\theta,z,\mathcal{M}_d}$; then z is frozen and each module is trained with the distillation term alone.

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 trained jointly.

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 compare these options in Appendix E.

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. (2026), 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:

$$\tilde{K}^{(l)} = [K_d^{(l)};\ K_{\text{text}}^{(l)}],\ \tilde{V}^{(l)} = [V_d^{(l)};\ V_{\text{text}}^{(l)}]. \tag{2}$$

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.

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 the choice of retained channels, and are not input-dependent gates. This lets us optimize a structured sparsity pattern jointly or sequentially 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:

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

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: intuitively, 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. The pretrained parameters θ are frozen during training, we only optimize the shared mask and the domain-module parameters. This isolates our question: whether existing capacity can be relocated from the backbone into memory modules while retaining the behaviour of the original model. Domain training uses a 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. In KOFF: joint, 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. In KOFF: sequential, the first stage updates only the mask, with no module attached; the second stage freezes the mask and updates only $\mathcal{M}_d$. Both variants are given the same budget: one pass over the domain corpus in total. KOFF: joint uses it for a single joint pass, while KOFF: sequential splits the corpus evenly, so its pruning stage and its module-training stage each see half of it. Differences between the two thus reflect the training schedule and not the amount of data or compute.

4 Experiments

Models We use 3 pretrained base models: Qwen-2.5-3B, Qwen-2.5-14B (Qwen et al., 2025), and Llama-3.1-8B (Grattafiori et al., 2024). We additionally report Llama-3.1-8B-Instruct to test whether the decomposition carries over to an instruction-tuned model.

Domain partitioning We use two different types of domains. First, **topic domains**: we use Wikipedia-Topics (Ziadé, 2024), which partitions English Wikipedia articles into 6 topic domains: Mathematics, Science, Technology, History, Culture, and Geography. We sample 30,000 articles per domain, split each article into chunks of at most 512 tokens, and apply a 95/5 train/evaluation split (approximately 200M tokens in total). As a second partitioning strategy we use **language domains**. Specifically, we use Wikisource (Wikimedia Foundation) and select 6 languages with sufficient coverage: English, Spanish, French, German, Italian, and Portuguese. Each language is treated as a separate domain with its own memory module; documents are chunked in the same way, and the total volume of the training data is again roughly 200M tokens.

Evaluation We use two metrics: (i) per-domain held-out perplexity, and (ii) MMLU (Hendrycks et al., 2021) (5-shot) for broad factual knowledge; for language domains we use Global MMLU (Singh et al., 2025) restricted to the six training languages, each evaluated with its matched module. Some experiments in the Appendix additionally report BigBench-Lite (BBL) (Srivastava et al., 2023).

Baselines We evaluate both KOFF variants: KOFF: joint, which learns the mask and memory modules together, and KOFF: sequential, which first learns the mask, freezes it, and trains the modules on top. Both are trained using the same budget and we compare them against two baselines. *Pruning-only* learns a mask in the same way, but without memory modules. This allows us to test how much of the removed capacity the KOFF modules recover. Secondly, we use LLM-Pruner (Ma et al., 2023) with a recovery stage, an established structured pruning method, as an external reference point. For recovery, we use the same slice of Wikipedia used to train the other models.

Pruning and module configuration We prune the attention 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 use LoRA rank 128 and 512 KV memory tokens in each setting. Moreover, we found that attaching memory modules to higher layers works best, thus we always attach them to the last 16 transformer blocks. Finally, we train one memory module per domain, i.e. 6 in each setting, with each module adding roughly 1–3% of the base model's parameters. At inference, when only one module is loaded, its LoRA adapter are merged into the pruned weights, so only the KV memory adds parameters. Thus, the backbone with one module is about 90% of the original size at 12% sparsity and 83% at 20%. For parameter counts and hyperparameters, see Appendix A.

	Topic domains		Language domains	
Model	MMLU↑	PPL↓	Global MMLU↑	PPL↓
Full Llama-8B (unpruned)	0.667	7.93	0.541	9.70
LLM-Pruner	0.471	9.98	0.396	18.72
Pruning only	0.400	10.88	0.532	14.03
KOFF: sequential	0.592	9.13	0.534	11.95
KOFF: joint	**0.602**	**8.12**	**0.537**	**10.90**

Table 1: Topic- and language-domain results at 20% global sparsity, Llama-8B base model. Pruning only uses the same L_0 sparsification and distillation objective but no memory modules. KOFF: sequential learns a mask in a first stage and then trains the memory modules on top of it; KOFF: joint learns the mask and the modules together. All models are trained for one pass over the domain corpus.

5 RESULTS

We ask whether a pretrained LLM's capacity can be reorganized so that broadly useful knowledge stays in a shared backbone while specialized knowledge moves to separate modules. A necessary condition is that attaching the matched memory module to a sparse backbone recovers more of the full model's performance than the backbone alone. Thus, we test this on Llama-3.1-8B (§5.1), across model sizes (§5.2), and on an instruction-tuned model (§5.3), and we ablate the memory modules and test mixed domains (§5.4). Whether the recovered knowledge is domain-specific is examined in §6.

5.1 MAIN RESULTS

Our main setting is Llama-3.1-8B at 20% global sparsity.[2] We use two domain partitions (§4): topics (subject areas) and languages. The latter introduces a stronger distribution shift, as the modules must account for different token distributions as well as different content. We evaluate on MMLU for topics and Global MMLU for languages, and in both settings report perplexity on held-out domain text with the matched module attached, averaged across domains.

Table 1 shows the results. In the topic setting, pruning alone substantially degrades perfromance: MMLU drops from 0.67 to 0.40 and perplexity rises from 7.9 to 10.9. Both KOFF variants recover most of this loss. KOFF: joint reaches 0.60 MMLU and 8.12 perplexity. KOFF: sequential reaches 0.59 and 9.1 on MMLU and perplexity respectively. In the language setting, perplexity shows the same trend: pruning-only results in 14.0 against 9.7 for the unpruned model, while KOFF: joint and KOFF: sequential reduce this to 10.9 and 12.0 respectively. Global MMLU, on the other hand, seems largely insensitive to this pruning setting. Pruning-only already retains 98% of the full model's performance, and both KOFF variants close this gap even further. This result is consistent with the intended decomposition. The capacity removed from the backbone mainly affects language-specific modeling of the text, captured by perplexity, while the knowledge probed by Global MMLU, which is largely shared across languages, remains in the backbone. LLM-Pruner, the external baseline, scores worse than both KOFF variants in all four settings. Finally, on this model KOFF: joint outperforms KOFF: sequential across all settings, suggesting that deciding what to remove from the backbone benefits from knowing what the modules can recover.

5.2 KOFF ACROSS MODEL SIZES

We now test whether the results hold across model sizes. Specifically, we test Qwen-2.5-3B at 12% and Qwen-2.5-14B and Llama-3.1-8B at 20% sparsity. Note that these models differ in sparsity. This is because the acceptable sparsity ratio depends on model size. For the 3B model, we found that model quality gradually degrades up to around 12% sparsity, but then decreases sharply, with memory modules no longer compensating at 20%. The 8B and 14B models were more resilient to larger sparsity ratios up to 20%. Thus, to test non-trivial capacity relocation, we report the 3B model at 12% and the larger models at 20%. However, this makes absolute scores not directly comparable

[2]Unlike unstructured pruning, typically reported at 50% sparsity, structured pruning is considerably harder and is studied at lower sparsity levels.

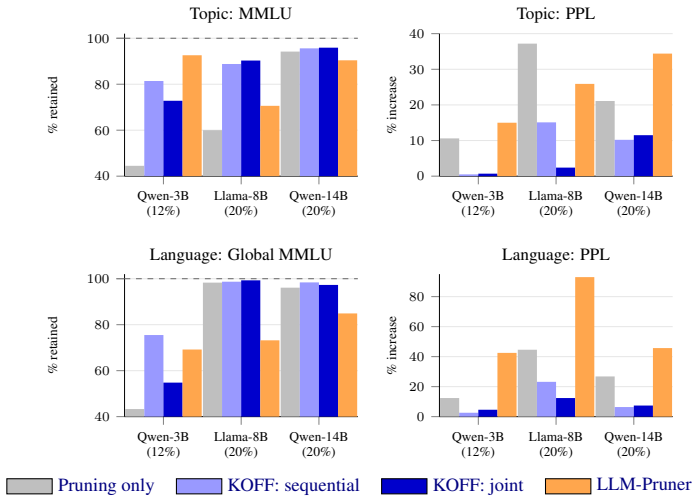

Figure 2: Results across model scales, relative to each model's unpruned version. For MMLU we show the percentage of the unpruned model's accuracy that is retained. For PPL we report the percentage increase in perplexity over the unpruned model (lower is better).

across models. Thus, in Figure 2 we report model performance relative to its own unpruned model version: the fraction of MMLU performance that is retained and the relative increase in perplexity.[3]

We find that KOFF improves over pruning-only perplexity for every model in both settings. Yet the extent of improvement seems largely dependent on how damaging pruning alone is. On Qwen-2.5-3B, pruning-only collapses MMLU to near random performance, even though the increase in perplexity stays modest. For this model, KOFF: sequential recovers 81% and 75% of unpruned MMLU performance. This is also the only setting where LLM-Pruner outperforms KOFF (on MMLU only), although at the cost of higher perplexity. On Qwen-2.5-14B, by contrast, pruning alone already retains 94–96% of MMLU and Global MMLU performance, leaving less room for recovery. Here KOFF gains are most prominent on perplexity evaluation. Finally, the preference between the two KOFF variants is model-dependent. KOFF: joint is better on Llama-3.1-8B. KOFF: sequential is better on Qwen-2.5-3B. On Qwen-2.5-14B the results are mixed, on MMLU KOFF: joint is slightly better and vice versa on Global MMLU, though differences are negligible. Across scales, the modules thus contribute most where pruning removes the most, when the backbone already retains most of the general capabilities, as in Qwen-2.5-14B, the modules mainly recover domain-specific modeling, reflected in perplexity rather than MMLU.

5.3 Instruction-tuned models

In Table 2, we report KOFF results on Llama-3.1-8B-Instruct at 20% sparsity. In the topic setting, we find that the results mirror those of the base model. KOFF improves over pruning-only, both in terms of MMLU and perplexity, with KOFF: joint performing best in both cases. For languages, both KOFF variants again substantially reduce perplexity, however, unlike the base model, they underperform on Global MMLU compared to pruning-only. Given that the language modules are trained on raw monolingual text, we hypothesize that they shift the instruction-tuned model toward plain continuation behavior, which helps perplexity but interferes with instruction-following capabilities in non-English languages. This also supports the hypothesis that the modules offload whatever the domain data represents, including its format. Thus, for instruction-tuned models, the memory modules might need to be trained on data that preserves the instruction format, which we leave to future work.

5.4 Ablations

Memory modules Each KOFF memory module combines two components: a LoRA adapter on the backbone weights and a learned KV cache. We first ablate them to test whether both are necessary,

[3]For results in absolute numbers, see Appendix B.

Method	Topic domains		Language domains	
	MMLU↑	PPL↓	Global MMLU↑	PPL↓
Unpruned	0.672	7.53	0.575	10.79
Pruning only	0.603	9.95	**0.531**	16.75
KOFF: sequential	0.618	9.37	0.445	**11.70**
KOFF: joint	**0.636**	**9.07**	0.459	12.21
LLM-Pruner	0.508	10.63	0.428	20.86

Table 2: Llama-3.1-8B-Instruct at 20% global sparsity.

	MMLU↑	PPL↓
Unpruned	0.67	7.9
Pruning only	0.40	10.9
Pruning + LoRA	0.56	8.9
Pruning + KV cache	0.58	8.5
KOFF: joint	**0.60**	**8.1**

Table 3: Memory module ablation on Llama-3.1-8B at 20% sparsity for topic domains. LoRA and KV cache are trained jointly during pruning.

	11 domains		3 domains	
	MMLU↑	PPL↓	MMLU↑	PPL↓
Unpruned	0.667	7.94	0.541	7.22
Pruning only	0.607	11.78	0.485	11.12
KOFF:seq.	**0.638**	10.68	**0.511**	10.10
KOFF: joint	0.630	**10.21**	0.504	**9.71**

Table 4: KOFF on Llama-3.1-8B at 20% sparsity using two mixed domain partitions: 11 domains contains data on all topics and languages (excl. English) and 3 domains includes Math, French and German.

training each component on its own jointly with the mask and comparing against the full module. We run this ablation on Llama-3.1-8B in the topic setting at 20% sparsity. In Table 3, we see that each component alone already recovers most of the loss from pruning. Pruning only drops MMLU from 0.67 to 0.40 and raises perplexity from 7.9 to 10.9. The KV cache alone brings these to 0.58 and 8.5, and perhaps surprisingly, is slightly better than LoRA alone on both metrics (0.56 and 8.9 respectively). However, combining the two gives the best result, retaining 90% of the original MMLU performance and increasing perplexity by only 2%.

Domain partitions In our main experiments, each partition contains 6 domains of a single category, either topics or languages. In practice, we may need to cover different types of domains at once, and the number of domains determines how much capacity each module must hold and how much the backbone must share across them. We therefore test whether the decomposition, and the preference for KOFF: joint on Llama-3.1-8B, holds under partitions that mix domain types and vary in size. Table 4 reports two such partitions on Llama-3.1-8B at $\sim$20% sparsity: 11 domains (6 topics and 5 languages) and a minimal mixed partition of 3 domains (Mathematics, French, German). Both KOFF variants improve over pruning only in both partitions: in the 11 domain mixture, perplexity drops from 11.78 to 10.68 (KOFF: sequential) and 10.21 (KOFF: joint); in the 3-domain mixture, from 11.12 to 10.10 and 9.71. The relative improvement over pruning only is nearly identical in both partitions (9% for KOFF: sequential and 13% for KOFF: joint), suggesting that a shared backbone can be learned across heterogeneous domains, and that the decomposition does not rely on domains being of a single type or on a particular number of domains.

6 Specialization analysis

In Section 5 we have demonstrated the effectiveness of KOFF in terms of recovering model performance. We now delve a bit deeper and test our hypothesis that the learned decomposition produces a separation of roles, i.e. the shared backbone retains general knowledge and the memory modules specialize. We first test to what extent memory modules are functionally specialized to their respective domains and we then look for structural specialization at the neuron level.

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

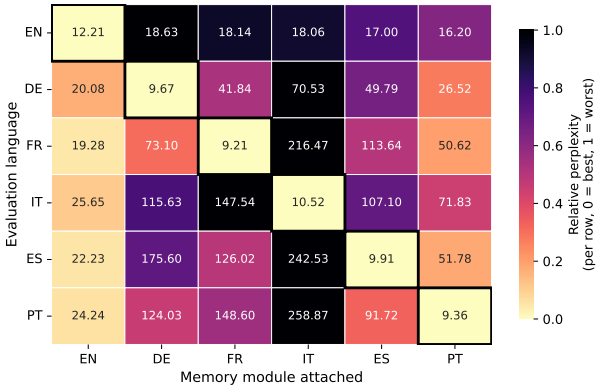

Figure 3: Cross-language memory-module swap on Llama-3.1-8B (KOFF: joint). We report perplexity scores when attaching the correct modules to the backbone (diagonal) versus mismatched ones.

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. For results on topic domains see Appendix B.

Language-specific capacity is preferentially offloaded Having shown that the modules specialize functionally, we ask whether the pruning mask shows a corresponding structural specialization. To do so, we use LAPE (Tang et al., 2024), a method that scores neurons based on how language-specific their activation patterns are. We run LAPE on the training corpus of each language separately and score the MLP neurons of Llama-3.1-8B. We then classify the 15% of neurons with the lowest entropy scores as language-specific and the 15% with the highest entropy as language-neutral. On Llama-3.1-8B model at 20% sparsity, we find that roughly 70% of the neurons that we classified as language-specific are removed from the backbone. In contrast, the language-neutral neurons are strongly protected: only 2.8% are removed, even though the relevant projections are pruned at roughly 40% local sparsity. Thus, KOFF: joint preferentially removes language-specific capacity while preserving language-neutral information. This is in line with our hypothesis that memory modules recover offloaded language-specific information.

7 Conclusion and Discussion

We introduced knowledge offloading, which decomposes a pretrained LLM into a shared sparse backbone and swappable memory modules, implemented as LoRA adapters and learned KV caches, and learned either sequentially or jointly. Across models and domains, attaching the matched module preserves much of the unpruned model's MMLU performance and substantially reduces the perplexity increase caused by pruning, compared to a sparse backbone without modules. This suggests that the capacity of a pretrained LLM can be reorganized into a shared core and separately managed specialized modules. Our specialization analyses further support this idea: matched modules work best, and language-specific neurons are preferentially removed while language-neutral neurons are protected.

Yet, what is preserved depends on what the modules are trained for. Our objective distills token-level distributions on raw domain text, which is reflected in perplexity and MMLU but not sufficient for generative behavior. In preliminary experiments, short-form generation was less well preserved for all pruned variants, including those with memory modules, and language modules hurt Global MMLU on the instruction-tuned model. Preserving richer behavior such as generation, reasoning traces, tool use, and safety-relevant responses likely requires training objectives and data that explicitly target these behaviors, which we leave for future work.

AI USE STATEMENT

Of the tasks on ICLR's required-disclosure list, generative AI was used only for implementing methods. All AI-assisted code was reviewed and tested by the authors, and every reported number was produced by running that code. No other task on the list was AI-assisted, including proposing or refining hypotheses, designing or giving feedback on experiments, and interpreting results.

Of the recommended-disclosure tasks, we used generative AI to edit author-written text for readability and to write auxiliary scripts for plotting and data processing. No citation, reference, or empirical claim was generated by an LLM.

We have reviewed all AI-assisted work and take responsibility for the final content of this paper, including any text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Saleh Ashkboos, Maximilian Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns. In *International Conference on Learning Representations*, volume 2024, pp. 11682–11701, 2024.

Rochelle Choenni, Ekaterina Shutova, and Dan Garrette. Examining modularity in multilingual lms via language-specialized subnetworks. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 287–301, 2024.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018. URL https://arxiv.org/abs/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. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Atri Rudra, James Y Zou, Azalia Mirhoseini, and Christopher Re. Cartridges: Lightweight and general-purpose long context representations via self-study. In *International Conference on Learning Representations*, volume 2026, pp. 42642–42687, 2026.

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

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, and Others. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Hanyu Hu, Pengxiang Zhao, Ping Li, Yi Zheng, Zhefeng Wang, and Xiaoming Yuan. Fasp: Fast and accurate structured pruning of large language models, 2025. URL https://arxiv.org/abs/2501.09412.

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. Sparse upcycling: Training mixture-of-experts from dense checkpoints. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=T5nUQDrM4u.

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

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3045–3059, 2021. URL https://aclanthology.org/2021.emnlp-main.243/.

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. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020. URL https://arxiv.org/abs/2005.11401.

Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A. Smith, and Luke Zettlemoyer. Branch-train-merge: Embarrassingly parallel training of expert language models. In *First Workshop on Interpolation Regularizers and Beyond at NeurIPS 2022*, 2022. URL https://openreview.net/forum?id=SQgVgE2Sq4.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing (volume 1: Long papers)*, pp. 4582–4597, 2021.

Jessy Lin, Luke Zettlemoyer, Gargi Ghosh, Wen-Tau Yih, Aram Markosyan, Vincent-Pierre Berges, and Barlas Oğuz. Continual learning via sparse memory finetuning. *arXiv preprint arXiv:2510.15103*, 2025.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1Y8hhg0b.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. LLM-pruner: On the structural pruning of large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=J8Ajf9WfXP.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. Mad-x: An adapter-based framework for multi-task cross-lingual transfer. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 7654–7673, 2020.

Jonas Pfeiffer, Sebastian Ruder, Ivan Vulić, and Edoardo Ponti. Modular deep learning. *Transactions on Machine Learning Research*, 2023.

Hadi Pouransari, David Grangier, C Thomas, Michael Kirchhof, and Oncel Tuzel. Pretraining with hierarchical memories: separating long-tail and common knowledge. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=XOu5z16cbY.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Shivalika Singh, Angelika Romanou, Clémentine Fourrier, David Ifeoluwa Adelani, Jian Gang Ngui, Daniel Vila-Suero, Peerat Limkonchotiwat, Kelly Marchisio, Wei Qi Leong, Yosephine Susanto, et al. Global mmlu: Understanding and addressing cultural and linguistic biases in multilingual evaluation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 18761–18799, 2025.

Aarohi Srivastava, Abhinav Rastogi, and Others. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=uyTL5Bvosj. Featured Certification.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/8fb21ee7a2207526da55a679f0332de2-Paper.pdf.

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. Branch-train-mix: Mixing expert llms into a mixture-of-experts llm. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=nqLAuMOF6n.

Tianyi Tang, Wenyang Luo, Haoyang Huang, Dongdong Zhang, Xiaolei Wang, Wayne Xin Zhao, Furu Wei, and Ji-Rong Wen. 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)*, pp. 5701–5715, 2024.

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

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. MoEfication: Transformer feed-forward layers are mixtures of experts. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 877–890, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.71. URL https://aclanthology.org/2022.findings-acl.71/.

Linxi Zhao, Sofian Zalouk, Christian K Belardi, Justin Lovelace, Jin Peng Zhou, Ryan Thomas Noonan, Dongyoung Go, Kilian Q Weinberger, Yoav Artzi, and Jennifer J Sun. Pre-training limited memory language models with internal and external knowledge. *ICLR*, 2026.

Tarek Ziadé. Wikipedia-topics dataset. Hugging Face Datasets, 2024. URL https://huggingface.co/datasets/tarekziade/wikipedia-topics. Wikipedia articles partitioned into 40 root categories.

A Implementation details

Table 5 lists the hyperparameters used in all main-text experiments. LoRA weights, gate parameters and KV memories use separate AdamW (Loshchilov & Hutter, 2019) optimizers. Models are loaded in bfloat16, and the 8B models use gradient checkpointing to fit in 80 GB of GPU memory. The 8B and 14B models use a lower λ, to account for their larger number of gates, and a batch size of 1.

Table 6 gives parameter counts. With one module attached, the model is 10–18% smaller than the original; the backbone together with all six modules is 88–107% of the original size.

The Llama-3.2-3B design experiments in Appendices D and E additionally use a small auxiliary term that periodically distills the teacher on general-domain text (C4 (Raffel et al., 2020), GSM8K (Cobbe et al., 2021) and ARC-Challenge (Clark et al., 2018)), updating only the mask. This term is not used in any main-text experiment.

Parameter	Value
LoRA rank r / scaling α	128 / 64
LoRA target modules	K, V, Gate, Down, Up
LoRA learning rate η_{LoRA}	3×10^{-4}
Gate learning rate η_{gate}	2×10^{-2}
Memory KV cache learning rate η_{mem}	1×10^{-2}
L0 λ (with auto-scaling)	25 (60 for 3B)
Temperature β	annealed from 2.0 to 0.5
Memory tokens N_{kv}	512 per domain
Memory-module layers	last 16 blocks
Batch size	4 (1 for 8B or larger)
Max sequence length	512
Epochs	1

Table 5: Default hyperparameters used in main-text experiments

		Memory modules		Merged model		Backbone + all modules	
Model	Base model	All domains	One domain	12%	20%	12%	20%
Qwen2.5-3B	3.09B	563.1M	93.8M	2.76B	2.54B	3.32B	3.10B
Llama-3.1-8B	8.03B	906.0M	151.0M	7.21B	6.65B	8.10B	7.54B
Qwen2.5-14B	14.77B	950.0M	158.3M	13.20B	12.14B	14.13B	13.07B

Table 6: Parameter counts of the base models, the KOFF memory modules, and the merged model used at inference. The merged model is the pruned backbone with one domain's module attached: the LoRA adapter is merged into the remaining weights and adds no inference-time parameters, while the KV cache is kept. Llama-3.1-8B-Instruct has the same architecture as Llama-3.1-8B, so its counts are identical.

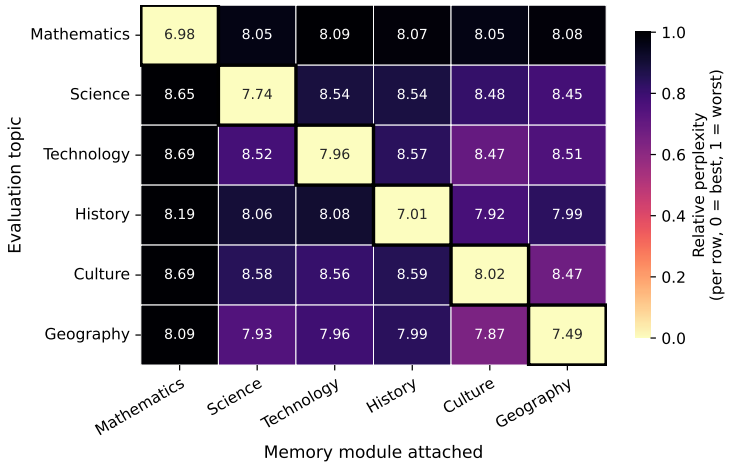

Figure 4: Cross-topic memory-module swap on Llama-3.1-8B (∼12% global sparsity). Each cell reports held-out perplexity on the row's evaluation topic when the column's memory module is attached to the shared backbone; the boxed diagonal is the matched module

A.1 Domain classification model for inference routing

We train a RoBERTa (Liu et al., 2019) 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. On a held-out test set the router's accuracy is 0.9675.

		Topic domains		Language domains	
Model	Sparsity	MMLU↑	PPL↓	Global MMLU↑	PPL↓
Qwen-3B (unpruned)	0%	0.633	8.21	0.571	13.12
Qwen-3B (LLM-Pruner)	12%	0.586	9.44	0.395	18.69
Qwen-3B (pruning only)	12%	0.282	9.08	0.247	14.75
Qwen-3B (KOFF: sequential)	12%	0.515	8.25	0.431	13.46
Qwen-3B (KOFF: joint)	12%	0.461	8.27	0.313	13.72
Llama-8B (unpruned)	0%	0.667	7.93	0.541	9.70
Llama-8B (LLM-Pruner)	12%	0.555	8.81	0.475	14.00
Llama-8B (pruning only)	12%	0.599	10.06	0.517	11.46
Llama-8B (KOFF: sequential)	12%	0.598	8.79	0.540	11.36
Llama-8B (KOFF: joint)	12%	0.590	8.60	0.537	10.91
Llama-8B (LLM-Pruner)	20%	0.471	9.98	0.396	18.72
Llama-8B (pruning only)	20%	0.400	10.88	0.532	14.03
Llama-8B (KOFF: sequential)	20%	0.592	9.13	0.534	11.95
Llama-8B (KOFF: joint)	20%	0.602	8.12	0.537	10.90
Llama-8B-I (unpruned)	0%	0.672	7.53	0.575	10.79
Llama-8B-I (LLM-Pruner)	20%	0.508	10.63	0.428	20.86
Llama-8B-I (pruning only)	20%	0.603	9.95	0.531	16.75
Llama-8B-I (KOFF: sequential)	20%	0.618	9.37	0.445	11.70
Llama-8B-I (KOFF: joint)	20%	0.636	9.07	0.459	12.21
Qwen-14B (unpruned)	0%	0.779	6.26	0.741	8.33
Qwen-14B (LLM-Pruner)	20%	0.704	8.41	0.629	12.14
Qwen-14B (pruning only)	20%	0.734	7.58	0.712	10.56
Qwen-14B (KOFF: sequential)	20%	0.745	6.90	0.729	8.86
Qwen-14B (KOFF: joint)	20%	0.747	6.98	0.721	8.95

Table 7: Topic- and language-domain results at 12% and 20% global sparsity. Pruning only uses the same L_0 sparsification and distillation objective but no memory modules. KOFF: sequential learns a mask in a first stage and then trains the memory modules on top of it; KOFF: joint learns the mask and the modules together. All systems are trained for one pass over the domain corpus in total. Global MMLU covers the six training languages, each evaluated with its matched module. Qwen-3B is reported only at 12% sparsity, since at 20% the memory modules no longer compensate for pruning

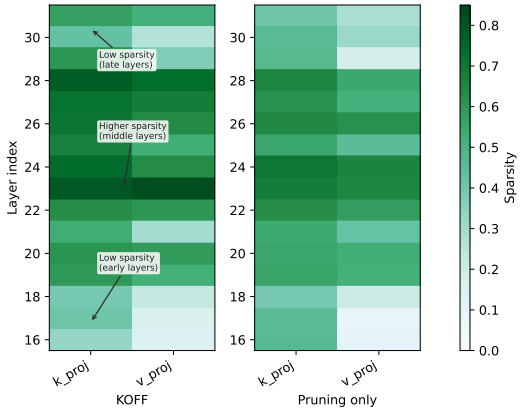

Figure 5: Per-layer sparsity for `k_proj` and `v_proj` layers of Llama-3.1-8B (language domains).

B FULL RESULTS

Table 7 gives the absolute scores behind Figure 2 and Table 2, together with Llama-3.1-8B at 12% sparsity. Figure 4 is the topic-domain counterpart of the cross-language module swap in Figure 3.

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

Table 8: Training-objective comparison on Llama-3.2-3B (topic domains, $\sim$12% global sparsity). Blended training diverges; teacher-only KL gives the best MMLU.

C 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.

D Distillation details

Table 8 compares three training objectives on Llama-3.2-3B with topic domains: next-token cross-entropy (SFT), KL to the teacher's top-100 logits (our default), and an equal mix of the two (α=0.5). KL gives higher MMLU and BBL than SFT at similar perplexity. The mixed objective diverges, as it also did in preliminary runs on Qwen-2.5-3B and Llama-3.1-8B; a possible cause is that the two terms conflict once pruning has shifted the student's distribution away from the teacher's.

E Memory module placement in the model and pruning targets

We study two design choices on Llama-3.2-3B with topic domains at a $\sim$12% global sparsity target: (a) which weight matrices the shared mask z acts on (Section E.1), and (b) which layers the memory modules are attached to (Section E.2).

E.1 Which Modules to Prune

On Llama-3.2-3B, the MLP projections (`gate`, `up`, `down`) hold roughly three quarters of the non-embedding weights, so gating only the attention projections cannot reach our 12% global sparsity target: `k,v` saturates at 2.2% and `q,k,v` at 2.8%. Pruning the attention projections this hard is also destructive: even at 2–3% global sparsity, both configurations score worse than gating `k,v` together with the MLP at 12% (Table 9).

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 near-random, against 0.520 for the matched `k,v`+MLP setup of Table 9; 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, so we allow a longer schedule when the MLP is gated.

E.2 Which layers to attach memory modules to

A memory module 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

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 9: Module-set comparison on Llama-3.2-3B (topic-domain, rank 128, $\sim$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.

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 128, 512 memory tokens, $\sim$12% global sparsity) fixed. Restricting memory modules to the upper 16 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.

Around this configuration, varying the sparsity weight $\lambda \in \{40, 60, 80\}$ changes MMLU only marginally (λ=40 is slightly ahead of λ=80, as it lands at a marginally lower sparsity), so layer placement remains the dominant design choice.