DYNAMIC EXPERT PRUNING FOR MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-Experts (MoE) architectures scale language models efficiently by activating only a few experts per token, but the saving is confined to computation: every expert must stay resident on the accelerator, so memory bounds where these models can be deployed. Expert pruning reduces this footprint, yet existing methods are static — a single mask, calibrated offline, is applied to the model for every subsequent request. This assumption can fail when the workload is heterogeneous, most prominently in multi-agent systems, where one backbone serves many tasks and roles at once: our analysis shows that different tasks and roles recruit different experts, while static methods assign one fixed subset to all of them. We therefore propose *Dynamic Expert Pruning* (DEP), which rests on a finding we establish here: an agent's system and task prompts are by themselves sufficient to identify the experts that agent and its task require, since that text already describes what the agent will do. A lightweight predictor, trained once on workflow transcripts, turns those prompts into a specialized per-request mask in a single forward pass, with no per-configuration calibration. Across diverse tasks and roles, model scales, and MoE architectures, DEP achieves better overall accuracy than static pruning and merging baselines, and generalizes to workflows unseen in training without retraining. Its margin over those baselines is largest when few experts are retained, suggesting that the role specialization inherent to multi-agent systems permits sparser serving than static pruning allows.

1 INTRODUCTION

Scaling language models has increasingly meant scaling them sparsely rather than densely. A Mixture-of-Experts (MoE) layer replaces a single feed-forward network with a pool of expert networks and a router that activates only a few of them per token, so the parameter count can grow without a proportional growth in per-token computation (Shazeer et al., 2017; Fedus et al., 2022). The design now underpins the strongest open models (Jiang et al., 2024; DeepSeek-AI, 2024; Yang et al., 2025) and frontier proprietary systems alike (Gemini Team, Google, 2024). The saving it delivers, however, is in arithmetic, not memory: routing is re-decided at every token, so any expert may be needed at any step and the entire pool must stay resident on the accelerator — or be paged in at a latency cost. Memory, not compute, therefore often determines where an MoE model can be deployed. This asymmetry has made expert pruning an active line of work — removing the experts that contribute least (Lu et al., 2024; Lasby et al., 2026), or merging redundant ones (Chen et al., 2025) — with a shared, encouraging finding: a large fraction of the experts can be discarded at modest loss, *provided* the retained subset matches the workload the model will actually serve.

Current methods simply assume that condition holds. Previous expert pruning methods are calibration-based: expert importance is averaged over a corpus taken to stand for future traffic, and the resulting *static* mask is frozen for every subsequent request. But a single backbone increasingly serves many prompting contexts at once — most prominently multi-agent systems, which assign it different tasks and agentic roles through prompts alone. Static calibration answers this diversity by finding one uniform, best-on-average set of experts; we argue that different tasks and roles may require different experts, so a single averaged subset ends up fitted to none of them.

Our analysis bears this out. Static pruning methods that deliver on single-workload benchmarks lose much of their accuracy in settings that mix tasks and roles, multi-agent systems chief among them,

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.

and the loss has a mechanism: agents in different roles and on different tasks recruit measurably different experts, so preserving quality calls for an expert mask *specialized* to each agent and task. Specialization by calibration, however, scales poorly — one pass per configuration, redone whenever a prompt is edited, a role is added, or a workflow is re-searched. This motivates expert pruning that is *dynamic*: specialized to each request, without a calibration pass for each.

This is precisely what we propose: **Dynamic Expert Pruning (DEP)**. Instead of calibrating a mask, DEP predicts it, with a lightweight predictor that learns which experts an agent on a given task will need from nothing but the agent's system prompt and its task input. A central claim of this paper is that *these prompts are by themselves sufficient*: the system prompt specifies what the agent is and what it will do, and the task input specifies what knowledge doing it demands, so the text that configures an agent is enough on its own to identify its specialized expert set before a single token is generated — and our experiments bear this out, on agents and workflows the predictor never saw. The predictor is trained on transcripts of the backbone at work — supervised by which experts contributed to each output, and by how little the masked model's distribution moves — and at serving time the mask is held fixed for the turn, preserving what makes static pruning deployable: a bounded resident expert set, reuse across a batch, untouched weights. Because it conditions on prompt text rather than a roster identifier, even agents and workflows never seen in training receive their masks *in one shot* — a single small forward pass on the system and task prompts, with no additional calibration.

Empirically, DEP achieves better overall accuracy than static pruning and merging baselines across benchmarks, workflows, model scales, and MoE architectures, and it generalizes to configurations unseen in training; our analyses trace why. No single static criterion is uniformly worse — each wins somewhere — but each also collapses somewhere, and DEP is the only policy that does not. Its advantage is largest in the sparsest regime: as the budget tightens, a fixed subset must cover every context and ends up serving none well, while DEP spends the whole budget on the experts one request needs. These results suggest that specialization opens the door to sparser pruning — narrowing a mask to a single agent's task and knowledge can let far fewer experts suffice.

Contributions.

- We analyze why static expert masks underperform in multi-agent systems — agents in different roles and on different tasks recruit different experts, so any single mask is fitted to none of them — motivating dynamic, per-agent expert masks.
- We show that an agent's system and task prompts are sufficient to identify the experts its task and role require, and build DEP on that finding: a mask predictor that reads only those prompts and emits a specialized expert mask on demand, *in one shot*, with no per-configuration calibration.
- We validate DEP across six agentic workflows, unseen structures, families, benchmarks, and domains, on SWE-bench Verified at 30B and 480B scale, and on a different MoE architecture, where it achieves the best overall accuracy among pruning policies and separates from them by a large margin in the sparsest regime — evidence that per-agent specialization can enable sparser serving.

2 Background and Motivation

Static expert pruning scores each expert on a calibration corpus — by routing frequency, reconstruction error (Lu et al., 2024), or router-weighted activation magnitude (Lasby et al., 2026) — and freezes the resulting subset, or a merged version of it (Chen et al., 2025), for every future request; the implicit assumption is that the corpus represents the traffic. Figure 1 shows what happens when it does not. With the backbone, retention, and calibration data held fixed, the same static masks that track the dense model when serving a single agent lose a large fraction of its accuracy on a multi-agent workflow over the same tasks, while a mask chosen per agent does not. The reversal matters because the workflow is there to buy accuracy in the first place: unpruned, the multi-agent system scores above the single agent, yet under the same static masks it ends up *below* it — static pruning does not merely cost accuracy here, it cancels the advantage the workflow was adopted to gain. The methods did not get worse; the workload changed underneath them — one subset must

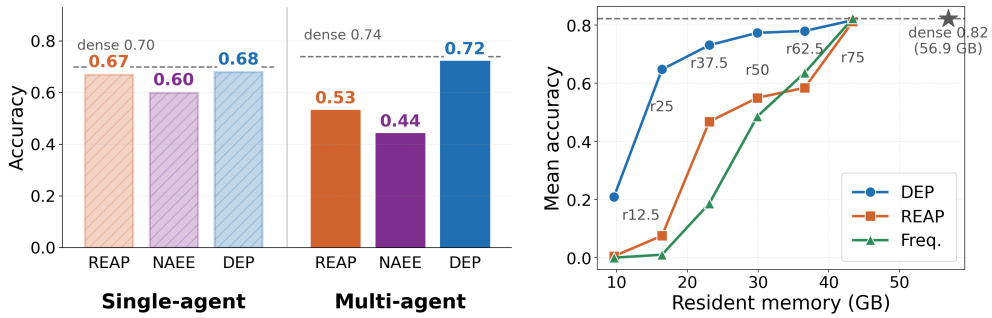

Figure 1: **Static masks that track the dense model for a single agent lose much of that accuracy on a multi-agent workload. Left:** identical pruning methods, backbone, retention (ρ=50%), and calibration data; only the serving setting changes. **Right:** the resulting accuracy–memory frontier.

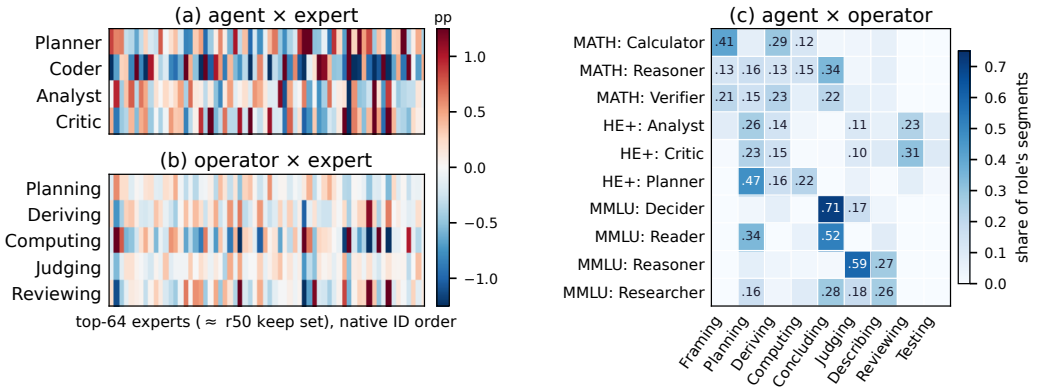

Figure 2: **Agents differ in what they do, and therefore in the experts they use. (a)** Per-role and **(b)** per-operator expert usage (HumanEval+, layer 24; deviation from the row mean over the top-64 experts, percentage points); **(c)** share of each role's segments per operator, across three tasks.

now serve many different roles and tasks at once, and it is fitted to none of them — and the failure is ungraceful: routing through the wrong experts degenerates into repetition rather than merely answering worse (Section 4.5). Whether multi-agent compositions outperform a single well-prompted model is actively debated (Cemri et al., 2025; Kim et al., 2025; Jwalapuram et al., 2026); nothing here depends on the answer, only on the observation that requests reaching one backbone now carry many different roles and instructions.

Figure 2 shows why. To describe agent behavior independently of agent identity, we segment agent outputs into nine *discourse operators* — Framing, Planning, Deriving, Computing, Concluding, Judging, Describing, Reviewing, and Testing — using a segmentation methodology adapted from Lee et al. (2026) (Appendix B). **Panel (c):** *different agents generate different operators* — a Planner spends nearly half its segments Planning, a Decider is dominated by Concluding, and mathematical and knowledge roles never Review or Test at all. **Panel (b):** *different operators use different experts* — the operator-conditional expert distributions differ on all nine tasks we measured, at 2.2–19× a permutation null that shuffles operator labels within each agent ($p \leq .006$), so the effect is not agent identity in disguise. **Panel (a):** *different agents use different experts*, the composition of the two. A prefill control supports the direction of causation: on identical token text, agents do not diverge — the prompt acts by changing what the agent writes, and what it writes determines where it routes. This is also what makes prediction possible: because the prompt determines the behavior and the behavior determines the routing, the prompt carries the information needed to anticipate an agent's experts before generation begins — the intuition DEP is built on.

This is the requirement the rest of the paper answers: agents in different roles and on different tasks call for *specialized* expert masks — Section 4.4 shows that masks of two agents in one workflow can overlap no more than masks of two unrelated benchmarks — and since calibrating one mask

per configuration reinstates the very cost pruning was meant to remove, the specialization must be computed from the agent's prompt rather than measured from a corpus. That is the method of Section 3. Appendix A situates this position among static compression, input-conditioned sparsity, and MoE serving systems.

3 METHODS

3.1 PROBLEM FORMULATION

We consider serving a frozen Mixture-of-Experts (MoE) language model $\mathcal{M}$ with L layers, each containing E routed experts of which the router activates the top-k per token. The workload is not a single task but a collection of multi-agent workflows $\{\mathcal{W}_1, \ldots, \mathcal{W}_M\}$: each workflow $\mathcal{W} = \{a_1, \ldots, a_n\}$ is a program over agents, where agent a is defined by a system prompt p_a and is invoked one or more times while the workflow processes an input problem q. All agents of all workflows are served by the backbone $\mathcal{M}$; they differ only in their prompts and intermediate context. Crucially, the set of workflows is open: workflows are edited, re-searched, and added over the life of a deployment, so a serving-time method cannot assume it has seen every configuration it will serve.

Expert pruning restricts each MoE layer ℓ to a subset $\mathcal{S}_\ell \subseteq \{1, \ldots, E\}$ of size K, the *retention* $\rho = K/E$. Static methods choose a single $\{\mathcal{S}_\ell\}_{\ell=1}^{L}$ offline from calibration data and apply it to every request of every workflow. DEP replaces the calibrated subset with a learned masking function

$$m_\theta(p_a, q) \;\in\; \{0,1\}^{L \times E}, \qquad \textstyle\sum_e m_\theta(p_a, q)_{\ell e} = K \;\; \forall \ell, \tag{1}$$

computed once before an agent's turn from the text that defines the request — a mask *specialized* to that request — and held constant for the whole turn.

3.2 ONE-SHOT MASK PREDICTION

DEP's *mask predictor* f_θ, orders of magnitude smaller than the backbone, maps the pair (p_a, q) to per-layer expert scores — operationalizing the intuition of Section 1 that the prompts configuring an agent already specify what it will do and what knowledge that demands, and therefore suffice as the mask's sole input. Following the sparsity-predictor design of instruction-following pruning (Hou et al., 2025), f_θ consists of a small pretrained text encoder and a shallow MLP head: the concatenated text $p_a \oplus q$ is encoded, the final-token hidden state $h \in \mathbb{R}^d$ is pooled, and the head produces scores

$$S \;=\; W_2\, \phi(W_1 h) \;\in\; \mathbb{R}^{L \times E}, \tag{2}$$

with ϕ a pointwise nonlinearity and the head's LE-dimensional output reshaped to $L \times E$. The mask of Eq. 1 keeps the top-K experts per layer by thresholding at the K-th largest score:

$$m_{\ell e} \;=\; \mathbf{1}[S_{\ell e} \geq \tau_K(S_\ell)]\,, \qquad \tau_K(S_\ell) = K\text{-th largest entry of } S_\ell. \tag{3}$$

Prediction is a single forward pass of the small encoder per (agent, problem), amortized over the agent's entire generation. Because the predictor reads prompt text rather than an identifier drawn from a fixed roster, nothing ties it to the agents it was trained on: any new prompt yields a mask, in one shot.

Applying the mask. The mask is installed on the frozen backbone at the router: logits of masked experts are set to $-\infty$ before the router's softmax-and-top-k, so routing renormalizes over the K kept experts and never selects a pruned one. Per-token top-k routing is otherwise unchanged, model weights are untouched, and we require $K \geq k$ so that routing remains well defined. This routing-level view is what accuracy evaluations use; the memory mechanism that realizes the saving is the serving runtime of Section 3.4.

3.3 PREDICTOR TRAINING

The predictor is trained on *workflow transcripts*: each training workflow is executed by the dense backbone on a calibration split disjoint from evaluation, and every agent call contributes a triple (system prompt p_a, input q, response r) attributed to the node that produced it. The resulting corpus pools all training workflows, so one predictor learns from the union rather than one artifact per

configuration. The backbone stays frozen; only θ is trained. Two complementary signals supervise the scores: one says which experts *mattered*, the other measures what a candidate mask *breaks*.

Impact-coverage target. For each transcript we compute a per-example expert-saliency tensor with the REAP criterion (Lasby et al., 2026): for layer ℓ and expert e,

$$\sigma_{\ell e} \;=\; \frac{1}{|\mathcal{T}_{\ell e}|} \sum_{t \in \mathcal{T}_{\ell e}} g_{\ell e}(x_t)\, \big\|\mathrm{FFN}_{\ell e}(x_t)\big\|_2, \tag{4}$$

where $\mathcal{T}_{\ell e}$ is the set of response tokens routed to e and $g_{\ell e}(x_t)$ the router gate weight — i.e. how much the expert actually *contributes* to this example's output, not merely how often it is routed. Binarizing σ per layer at top-K gives a per-example target mask $m^\star$, and the predictor is trained with binary cross-entropy, $\mathcal{L}_{\text{impact}} = \mathrm{BCE}(\mathrm{sigmoid}(S), m^\star)$. This target is cheap and per-example, but it scores experts in isolation and never sees the joint effect of removing $E{-}K$ of them at once.

Self-distillation. The second signal closes that gap by measuring masking damage directly: with teacher the unmasked backbone and student the same backbone under the predicted mask,

$$\mathcal{L}_{\text{KD}} \;=\; \frac{1}{|\mathcal{R}|} \sum_{t \in \mathcal{R}} \mathrm{KL}\big(P_{\mathcal{M}}(\cdot \mid x_{<t}) \;\big\|\; P_{\mathcal{M}\odot\tilde{m}}(\cdot \mid x_{<t})\big)\,, \tag{5}$$

averaged over the response tokens $\mathcal{R}$ of the transcript. The teacher is the model itself, so no labels are needed and the loss directly optimizes the quantity deployment cares about — how little the masked model's predictive distribution moves. To make Eq. 5 differentiable in θ, the hard top-K of Eq. 3 is relaxed to a per-layer soft mask $\tilde{m}_{\ell e} = \mathrm{sigmoid}((S_{\ell e} - \tau_K(S_\ell))/T)$ whose temperature T is annealed toward the hard mask over training; at inference we use the hard top-K of the same scores. The total objective mixes the two signals,

$$\mathcal{L} \;=\; \alpha\, \mathcal{L}_{\text{KD}} \;+\; (1-\alpha)\, \mathcal{L}_{\text{impact}}, \tag{6}$$

with mixing weight α. One predictor is trained per retention level on the pooled corpus; the same checkpoint then serves every agent of every workflow at that retention, seen or unseen.

3.4 Dynamic inference and delta loading

Inference. A request arrives as (agent prompt, input). The serving engine first obtains the request's mask: one predictor forward if it is new, or a cache hit — masks are content-addressed by (p_a, q), so repeated calls of an agent on the same problem, and batches grouped by agent, reuse one mask. The backbone then decodes the entire turn under that mask: per-token top-k routing runs natively, restricted to the K kept experts. Nothing about the mask changes mid-turn, so the inner decoding loop is identical to serving a statically pruned model. Resident memory is the non-expert weights plus a ρ fraction of the experts — which dominate an MoE's footprint, so memory shrinks nearly in proportion to ρ — and adapting to a new workflow, agent, or edited prompt costs one small predictor forward rather than the recalibration pass a static mask requires.

Delta loading. The memory saving comes from keeping only the kept experts on the accelerator. Expert weights live as per-expert tensors in host memory; each layer holds a resident buffer of K slots plus an index map from global expert ids to slots, and the router is masked to the resident set so an absent expert can never be selected. When the mask changes — typically at an agent hand-off — the engine copies only the set difference between the required and resident experts into the slots that evicted ones vacate, and updates the map; experts shared by consecutive masks never move. That delta stays small because the masks of agents in one workflow overlap substantially (Section 4.4), and it is largely hideable because the workflow's topology is known in advance: the next agent's mask can be predicted and its delta prefetched while the current turn is still decoding, and batching by agent amortizes whatever remains.

4 Experiments

4.1 Experimental setup

Models and workflows. Our main backbone is Qwen3-30B-A3B (Yang et al., 2025) (48 MoE layers, $E{=}128$ routed experts, top-8 routing), evaluated at retention $\rho{=}50\%$ ($K{=}64$) and the ag-

Table 1: **DEP gives the best overall accuracy at both retentions, and is the only policy still usable at ρ=25%.** AFlow-discovered multi-agent workflows (Qwen3-30B-A3B); DEP against static masks calibrated on the *identical* union. Values are % (pass@1; HotpotQA SQuAD-F1; MMLU accuracy), mean ± std; protocol in Section 4.1.

Method	MATH-500	GSM8K	HumanEval+	MBPP+	MMLU	HotpotQA	*Mean*
Dense (56.9 GB)	68.3 ±8.6	97.3 ±1.2	92.8 ±1.4	80.0 ±6.9	82.7 ±1.5	71.9 ±1.7	82.2 ±1.0
50% retention (keep 64/128; 29.9 GB resident)							
Freq.	62.3 ±9.0	**95.3** ±1.5	2.9 ±1.4	3.0 ±1.0	51.0 ±2.6	**65.4** ±3.7	46.7 ±1.9
REAP	40.7 ±6.8	71.3 ±4.5	78.7 ±0.8	45.7 ±4.0	40.3 ±4.0	47.3 ±3.0	54.0 ±1.6
HC-SMoE	60.3 ±6.4	84.0 ±3.0	75.8 ±2.2	75.0 ±7.8	48.3 ±0.6	60.4 ±1.6	67.3 ±1.5
DEP	**67.3** ±10.6	**95.3** ±2.3	**84.5** ±0.8	**76.7** ±2.5	**70.7** ±4.7	60.7 ±3.2	**75.9** ±2.3
25% retention (keep 32/128; 16.4 GB resident)							
Freq.	4.0 ±1.0	2.3 ±0.6	0.0 ±0.0	0.0 ±0.0	0.7 ±1.2	0.2 ±0.0	1.2 ±0.2
REAP	5.3 ±1.5	7.7 ±4.0	0.0 ±0.0	12.3 ±2.1	10.7 ±2.5	4.3 ±0.9	6.7 ±1.4
HC-SMoE	2.7 ±1.5	0.7 ±1.2	0.0 ±0.0	0.3 ±0.6	26.3 ±1.5	5.0 ±0.6	5.8 ±0.1
DEP	**51.3** ±7.2	**87.7** ±4.7	**78.3** ±2.9	**61.7** ±6.5	**48.0** ±1.0	**53.7** ±2.7	**63.4** ±1.6

gressive ρ=25% (K=32); the predictor encoder is Qwen2-0.5B. For agentic coding (Section 4.6) we use Qwen3-Coder-30B-A3B-Instruct and, to test scale, Qwen3-Coder-480B-A35B-Instruct-FP8 (62 layers, E=160); to test a different MoE family we additionally use DeepSeek-V2.5-236B (59 layers, E=160 routed, top-6, plus 2 shared experts). Workflows are produced by automated search rather than hand-designed: per task, an AFlow MCTS search (Zhang et al., 2025) with the target backbone as executor, from which we select the best-scoring discovered workflow (Appendix B).

Tasks and protocol. Six tasks span mathematical reasoning, code generation, and knowledge-seeking QA: MATH-500 (Hendrycks et al., 2021b; Lightman et al., 2024), GSM8K (Cobbe et al., 2021), HumanEval+ and MBPP+ (Chen et al., 2021; Austin et al., 2021; Liu et al., 2023a), MMLU (Hendrycks et al., 2021a), and HotpotQA (Yang et al., 2018), each with one discovered workflow; n=100 held-out problems per seed over 3 seeds (HumanEval+ caps at 69). Evaluation pools are disjoint from search-validation, predictor-training, and calibration data. Agentic coding uses a fixed 50-instance subset of SWE-bench Verified (Jimenez et al., 2024; Chowdhury et al., 2024) under the official harness in an oracle-assisted setting, so the object of comparison is the *relative* ordering of policies, not leaderboard-absolute numbers.

Baselines. All policies share the identical frozen backbone and workflow, and every calibration-based baseline is calibrated on the *same* transcripts that train our predictor: **dense** (unpruned reference), **direct** (unpruned single call — the workflow's own contribution), **freq** (static mask by routing frequency), **REAP** (Lasby et al., 2026) (static mask by the saliency of Eq. 4, a state-of-the-art static pruning criterion), **HC-SMoE** (Chen et al., 2025) (expert *merging* by agglomerative clustering), and **DEP** (ours; the per-request predictor mask of Section 3). One predictor per retention is trained on the pooled six-workflow transcript corpus (∼1.5k examples balanced across tasks) with the objective of Eq. 6; accuracy runs mask routing only, while memory and latency are measured separately with the delta-loading runtime of Section 3.4. All remaining details — workflow structures, contamination checks, training hyperparameters, efficiency protocol, and hardware — are in Appendix B; ablations of the training recipe are in Appendix C.

4.2 Dynamic versus static expert pruning

Table 1 compares all policies on six benchmarks, each served by a workflow discovered for it by AFlow, at both retention levels.

DEP obtains the best overall accuracy of any pruning policy at both retention levels, and leads on most individual benchmarks. The more telling pattern, though, is not where the margins are but where the static baselines break down: each of them collapses on some subset of the workload while remaining respectable on the rest. We believe this is what a single mask costs. A static criterion must score every expert on a mixture of tasks and agents at once, so experts that matter to only part of that mixture look unimportant on average and are dropped — and whichever task depended on them fails outright rather than degrading. DEP never has to make that compromise: it emits a separate mask per agent and task, so an expert only has to matter to the request being served in order to be

Table 2: **One predictor, fixed after training, keeps working as the workload moves away from it.** Generalization ladder: one universal predictor and one static calibration, both fixed after training on a three-task workflow pool, evaluated at increasing distance from training. Accuracy % (HotpotQA F1), 30 problems per cell, mean ± std over 3 seeds. **Bold** = best, underline = second-best pruned method per row and retention.

			$\rho = 50\%$				$\rho = 25\%$			
	task	Dense	Freq.	REAP	HC-SMoE	DEP	Freq.	REAP	HC-SMoE	DEP
Unseen structure	HotpotQA	68.0±6.6	12.9±1.5	12.3±2.7	54.5±5.5	**65.0±2.8**	0.4±0.4	5.3±3.3	0.1±0.1	**38.3±0.6**
	HumanEval+	94.3±3.5	39.0±2.6	59.3±1.5	82.3±1.5	**83.4±4.1**	0.0±0.0	7.7±2.5	0.0±0.0	**9.7±3.2**
	MATH	68.3±3.3	0.0±0.0	2.8±2.5	45.8±8.2	**55.0±2.4**	0.0±0.0	0.0±0.0	0.0±0.0	**9.4±1.0**
Unseen family	HotpotQA	67.5±4.3	9.9±0.1	29.5±8.9	**56.4±7.2**	53.1±3.2	0.0±0.0	3.1±1.2	0.0±0.0	**44.4±2.6**
	HumanEval+	92.0±1.9	58.7±1.7	74.0±2.1	87.6±3.4	**88.2±3.9**	0.0±0.0	27.5±3.2	0.0±0.0	**28.0±2.3**
	MATH	68.3±0.8	30.0±1.3	2.2±1.4	48.9±2.5	**54.4±1.4**	0.0±0.0	1.1±0.6	0.0±0.0	**21.7±3.9**
Unseen benchmark	SVAMP	91.9±3.6	28.3±3.9	19.4±5.5	**87.8±3.1**	82.8±0.8	1.1±0.6	12.8±2.4	1.1±0.6	**51.7±5.5**
	MBPP	63.3±5.2	24.4±3.4	47.8±11.0	**51.9±3.6**	46.7±1.6	0.0±0.0	**27.2±5.5**	0.6±0.8	8.3±2.4
Unseen domain	MMLU	68.3±2.4	12.2±1.6	13.3±2.5	33.3±3.6	**48.9±3.1**	0.0±0.0	10.0±1.7	1.7±0.8	**15.6±1.6**

kept. That it can do so from the system and task prompts alone is the part we would emphasize — the text that configures an agent is enough to identify the experts that agent will need, before it has generated a token.

The same effect sharpens at ρ=25%, where DEP's margin over every baseline is at its largest. Halving the budget again leaves no room for a mask to hedge: a subset chosen to be adequate everywhere is now too small to be sufficient anywhere, and the static policies lose the experts their tasks actually depend on, which shows up as collapsed rather than merely weakened generation (Section 4.5). Spending the same budget on one agent's experts instead keeps what that agent depends on, and the kept set stays coherent even when it is small. DEP does give up real accuracy against dense here — this is graceful degradation, not free compression — but the regime is informative in its own right: it suggests that a multi-agent decomposition is itself a form of specialization that can be exploited for efficiency, since an agent responsible for one narrow part of a task needs correspondingly fewer experts, and can therefore be served far more sparsely than a model expected to do everything.

4.3 Generalization to unseen workflows and benchmarks

Section 1 claimed that DEP masks new configurations in one shot, because nothing ties a prompt-conditioned predictor to the workflows it was trained on. We test that claim directly with a single universal predictor. Unlike the other experiments, which serve one discovered workflow per task, here the predictor is trained on transcripts from a *pool* of 32 discovered workflows spanning three tasks (full rosters in Appendix D), then evaluated *without any retraining* on a ladder of increasing distance from training: **unseen structures** of workflow families seen in training; an entire **unseen workflow family**; **unseen benchmarks** served by unseen workflows (SVAMP (Patel et al., 2021), near the training math distribution; MBPP, far); and an **unseen domain** (MMLU). Every static baseline is calibrated once on the same training transcripts and reused unchanged at every rung — exactly the situation a static mask faces when the workload changes after calibration.

The predictor transfers. With no retraining and no recalibration it is the strongest policy on most rungs at both retentions (Table 2), and at ρ=50% it comes close to the dense reference on several of them: a mask produced for a workflow the predictor has never seen is still a mask worth serving. This is the claim of Section 1 tested at its weakest point — the prompts here describe agents, structures, and in two rungs whole benchmarks and domains that were absent from training — and the prompts still carry enough to pick a usable expert set.

Two things temper the result. Expert merging remains competitive while the budget is loose, leading on a minority of rungs at ρ=50% and on the unseen benchmarks in particular; averaging experts rather than committing to a subset is a reasonable hedge when the workload is unfamiliar and the budget can absorb it. And with 30 problems per cell the seed spread is not negligible: on a few rungs the gap between the best and second-best policy is smaller than a seed standard deviation, so the ordering across the ladder, rather than any individual cell, is what this table supports. At ρ=25%

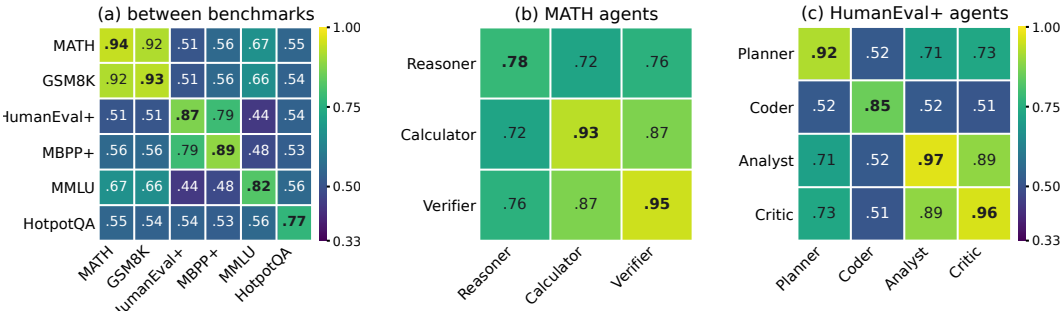

Figure 3: **Predicted masks organize themselves by task and by role.** Jaccard overlap of predicted kept-expert masks at ρ=50% (chance = 0.33). **(a)** Between benchmarks; **(b, c)** between the agents of the MATH and HumanEval+ teams. Diagonals: the same row on different problems. Measured on hand-crafted agent teams, not the discovered workflows of Table 1.

Table 3: **A wrong mask wastes more time rambling than a right mask spends swapping experts.** Efficiency at ρ=50% over three workflows (HotpotQA, HumanEval+, MATH; 3 random 50-problem seeds), not comparable with Table 1; per task in Table 12. Tokens and times are measured; Δ-load is an analytic *upper bound* on the per-agent-step swap cost, which static masks never pay.

policy	memory (GB)	accuracy	tokens/prob	generation (s)	Δ-load (s)	total (s)
dense	56.9	**0.78**	1905	3239	—	3239
DEP	**29.9**	0.76	**1872**	3275	+42	**3317**
REAP	**29.9**	0.53	2077	3803	0	3803
freq	**29.9**	0.27	3365	5457	0	5457

the picture sharpens: merging collapses, and DEP leads on every non-code rung by margins well outside the seed spread. Code is the exception, REAP stays competitive there, and on the unseen code benchmark it is the stronger policy. We read that contrast as a difference in what the two kinds of artifact can express: a calibrated mask answers "which experts does *this* workload need", a question with no answer once the workload changes, whereas the predictor has learned "which experts does an agent *like this* need", which still has an answer for a configuration it has never seen.

4.4 Analysis: predicted masks specialize by task and role

How does the predictor actually spend its budget? Figure 3 looks inside the predicted masks, measuring the overlap of the kept-expert sets between benchmarks and between the agents of one team. The setting is hand-crafted agent teams that follow the role decompositions in widespread use in multi-agent frameworks (Wu et al., 2024; Hong et al., 2024). The masks organize themselves by expertise. Between benchmarks, domains that share skills share experts: MATH and GSM8K keep largely the same set, as do HumanEval+ and MBPP+, while overlap across domain boundaries drops sharply. Between agents, the Coder keeps a visibly distinct expert set while the Analyst and Critic share most of theirs — mirroring the routed-usage picture of Figure 2(a), where the Coder was likewise the outlier role. None of this structure is supervised: the predictor is trained only with the objective of Eq. 6, yet it allocates experts that track what each task and role actually requires — the specialization that Section 2 argued multi-agent serving demands.

This heterogeneity is consistent with the pattern in Table 1, though it is measured on these teams rather than on the discovered workflows served there. Where even closely related requests keep overlapping-but-different sets, any single static subset misfits a large fraction of the request stream — harmlessly while the budget has slack, fatally at ρ=25%: a static mask is in effect an average of incompatible masks, and the sparser the budget, the more that averaging would cost.

4.5 Efficiency: memory, wall-time, and the failure mode of wrong masks

With delta loading (Section 3.4), resident memory shrinks nearly in proportion to the retention, and Figure 1 (right) traces the resulting accuracy–memory frontier: DEP holds accuracy at footprints where both static masks have already collapsed, reconverging with them only at mild compression,

Table 4: **The ordering carries over to agentic coding at both model scales.** SWE-bench Verified resolve rate (%) at ρ=50%.

Method	Coder-30B	Coder-480B
Dense (workflow)	40.0	56.0
Direct (single call)	12.0	50.0
Freq.	2.0	0.0
REAP	22.0	44.0
DEP	**30.0**	**50.0**

Table 5: **It also holds on a different MoE family.** DeepSeek-V2.5-236B, mean over five tasks (%); per task in Table 13.

Method	ρ=50%	ρ=25%
Dense (workflow)	70.7	
Direct (single call)	71.4	
Freq.	33.0	0.8
REAP	46.5	1.5
DEP	**51.7**	**6.9**

where pruning is nearly free for everyone. DEP's overheads — one predictor forward per request and an expert swap per agent hand-off — add about one percent to the measured generation wall in aggregate, and under ten percent on any single task, when charged at their analytic worst case (Table 3). Even with that upper bound applied to DEP and no swap cost at all charged to the static masks, DEP still finishes *faster* overall than they do. The reason is token waste: a mask that has lost essential experts does not answer more briefly, it degenerates, emitting repetitive tokens until the generation cap (Appendix G) — a wrong mask loses more time rambling than a right mask spends loading, so the accuracy, token, and latency gaps in Table 3 all share one cause.

4.6 Ablation analysis: scale and architecture agnosticism

Finally we ask whether the result depends on the setting that produced it, along three axes: a multi-agent benchmark outside the training suite, a much larger backbone, and a different MoE architecture. On SWE-bench Verified (Table 4) DEP is the strongest pruning policy at both model scales, on the 30B coder and on the 16× larger 480B one, whose dense entry is a floor because four instances hit harness image errors. The setting is oracle-assisted, so the ordering rather than the absolute rate is the result — and it is the same ordering Table 1 produced on a different task family. Even the failure mode recurs: the frequency mask does not go silent but emits degenerate patches, garbage diffs in place of garbage tokens.

Nothing in DEP's design is specific to one MoE family: the mask acts at the router, and the serving runtime assumes only per-expert weight tensors. Table 5 tests that on DeepSeek-V2.5-236B, a different MoE lineage with its own routing width and a pair of shared, always-active experts we leave unmasked. DEP again gives the best overall accuracy of any pruning policy at both retentions, and per task (Table 13) the frequency mask again breaks first on code. What does not transfer is the retention level: pruning costs more of the dense accuracy here than on Qwen3, and at ρ=25% every policy becomes unusable, DEP included, even though it keeps the largest margin. How aggressively a backbone tolerates pruning is thus a property of that backbone, to be measured; what carries over is which policy to use once the budget is fixed.

5 Conclusion

Serving many agents from one backbone breaks the assumption expert pruning rests on: agents in different roles and on different tasks recruit measurably different experts, so a mask averaged over all of them is fitted to none, and pruning has to become dynamic rather than better calibrated. DEP makes it dynamic by prediction rather than by calibration: a small predictor, trained once on workflow transcripts and frozen thereafter, emits a per-request mask in a single forward pass. The observation we would most want carried forward is that the prompts already suffice — the system and task prompts that configure an agent are by themselves enough to identify the experts it will need, before it has generated a token and even when neither the agent nor its workflow appeared in training. Our experiments bear that out across workloads, model scales and MoE architectures, and our analyses trace the behavior to its cause: masks that specialize by task and by role, and a wrong mask that costs more in degenerate generation than a right one costs in expert swaps. The price is a single predictor training run, and how aggressively a given backbone tolerates pruning must still be measured rather than assumed (Appendix H). Role specialization, then, cuts both ways: the division of labor that multi-agent systems adopt for capability also narrows what each agent must know, and a narrower need is one that far fewer experts can serve.

AI USE STATEMENT

Generative AI was used as a writing and coding assistant: to help draft and edit prose. The research idea, the method, and the claims are solely the authors' own. All AI-assisted work was reviewed by the authors — every reported number was produced by our own runs and checked against the stored prediction files, and all code was read and tested before use. We take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

Section 3 specifies the masking function, training objective and serving procedure in full (Eqs. 1–6), and Appendix B gives workflow discovery, the predictor's training configuration and compute, the efficiency protocol and the hardware; Appendices D and C give the generalization rosters and the recipe's sensitivity. Two conditions matter most for comparability and are stated in Section 4.1: every calibration-based baseline is calibrated on the same transcripts that train our predictor, and evaluation pools are disjoint from search-validation, predictor-training and all calibration data, verified by text-overlap checks. Code for predictor training, the masking and delta-loading runtime, and the evaluation harness is provided as supplementary material.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

I-Chun Chen, Hsu-Shen Liu, Wei-Fang Sun, Chen-Hao Chao, Yen-Chang Hsu, and Chun-Yi Lee. Retraining-free merging of sparse MoE via hierarchical clustering. In *International Conference on Machine Learning (ICML)*, 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench verified. OpenAI Blog, `https://openai.com/index/introducing-swe-bench-verified/`, 2024.

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. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Artyom Eliseev and Denis Mazur. Fast inference of mixture-of-experts language models with offloading. *arXiv preprint arXiv:2312.17238*, 2023.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*, 2023.

Gemini Team, Google. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *NeurIPS Datasets and Benchmarks Track*, 2021b.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024.

Bairu Hou, Qibin Chen, Jianyu Wang, Guoli Yin, Chong Wang, Nan Du, Ruoming Pang, Shiyu Chang, and Tao Lei. Instruction-following pruning for large language models. In *International Conference on Machine Learning (ICML)*, 2025.

Ranggi Hwang, Jianyu Wei, Shijie Cao, Changho Hwang, Xiaohu Tang, Ting Cao, and Mao Yang. Pre-gated MoE: An algorithm-system co-design for fast and scalable mixture-of-expert inference. In *International Symposium on Computer Architecture (ISCA)*, 2024.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024.

Prathyusha Jwalapuram, Hehai Lin, Chuyuan Li, Fangkai Jiao, Sudong Wang, Yifei Ming, Zixuan Ke, Chengwei Qin, Giuseppe Carenini, and Shafiq Joty. The illusion of multi-agent advantage. *arXiv preprint arXiv:2606.13003*, 2026.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, Mark Malhotra, Paul Pu Liang, Hae Won Park, Yuzhe Yang, Xuhai Xu, Yilun Du, Shwetak Patel, Tim Althoff, Daniel McDuff, and Xin Liu. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025.

Mike Lasby, Ivan Lazarevich, Nish Sinnadurai, Sean Lie, Yani Ioannou, and Vithursan Thangarasa. REAP the experts: Why pruning prevails for one-shot MoE compression. In *International Conference on Learning Representations (ICLR)*, 2026.

Daniel Lee, Owen Queen, and James Zou. ReasonOps: Operator segmentation for LLM reasoning traces. *arXiv preprint arXiv:2605.29192*, 2026.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024.

Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of LLM-based applications with semantic variable. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024a.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for LLM compression and acceleration. In *Conference on Machine Learning and Systems (MLSys)*, 2024b.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Ré, and Beidi Chen. Deja vu: Contextual sparsity for efficient LLMs at inference time. In *International Conference on Machine Learning (ICML)*, 2023b.

Xudong Lu, Qi Liu, Yuhui Xu, Aojun Zhou, Siyuan Huang, Bo Zhang, Junchi Yan, and Hongsheng Li. Not all experts are equal: Efficient expert pruning and skipping for mixture-of-experts large language models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple math word problems? In *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2021.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *First Conference on Language Modeling (COLM)*, 2024.

Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh Marina. MoE-Infinity: Offloading-efficient MoE model serving. *arXiv preprint arXiv:2401.14361*, 2024.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations (ICLR)*, 2025.

A Related work

Static MoE expert compression. Static expert pruning scores experts on a calibration corpus and freezes the result: by routing frequency, by the reconstruction error of removing an expert (Lu et al., 2024), or by router-weighted activation magnitude (Lasby et al., 2026), a state-of-the-art static criterion; HC-SMoE (Chen et al., 2025) instead *merges* redundant experts by agglomerative clustering, trading peak accuracy for graceful degradation. All of these produce a single artifact fitted to the calibration distribution — the assumption that Section 2 shows failing in multi-agent serving — and all serve as baselines in Section 4.

Input-conditioned sparsity. Closer to DEP are methods that condition sparsity on the input rather than on a corpus. Deja Vu (Liu et al., 2023b) exploits per-token contextual sparsity in *dense* models, predicting on the fly which attention heads and MLP neurons a token needs; the prediction is per token and per layer, so the resident weight set changes continuously. Instruction-following pruning (Hou et al., 2025), whose predictor design we adopt (Section 3.2), conditions a dense-model FFN mask on the instruction, but selects among sparsity patterns for a single-call setting and fine-tunes the backbone jointly with the predictor. DEP differs in granularity and setting: it masks *routed experts* of a frozen MoE backbone, holds the mask fixed for a whole agent turn so that the resident set is bounded and cacheable, is trained once on pooled multi-agent transcripts with a self-distillation objective (Eq. 5), and is evaluated on its central claim — one-shot masks for agents, workflows, and tasks never seen in training (Section 4.3) — which per-call sparsity selection does not address.

MoE offloading and serving systems. A complementary line keeps all experts but moves them: quantized expert offloading with LRU caching and speculative prefetch (Eliseev & Mazur, 2023), activation-aware request-level tracing for prefetching and caching (Xue et al., 2024), and pre-gating that computes the next layer's expert set one layer ahead (Hwang et al., 2024). These systems chase per-token routing decisions reactively; DEP inverts the problem — the whole turn's expert set is known *before generation begins*, so expert movement becomes a deterministic per-turn delta that can be prefetched at agent hand-offs (Section 3.4) rather than a per-token cache-miss stream. Weight quantization (Frantar et al., 2023; Lin et al., 2024b) reduces the *bytes per expert* rather than the number of resident experts; it is orthogonal to DEP and composable with it, as our FP8 480B experiments already illustrate.

Multi-agent systems and their efficiency. Multi-agent frameworks assign the same backbone differentiated roles through prompting (Wu et al., 2024; Hong et al., 2024), and automated search extends this by discovering workflow programs (Zhang et al., 2025); whether such compositions beat a single well-prompted model is actively debated (Cemri et al., 2025; Kim et al., 2025; Jwalapuram et al., 2026), but either way the resulting traffic is heterogeneous, which is all DEP requires. Systems work on serving LLM *applications* optimizes scheduling and cross-request state — e.g. exposing inter-request dependencies to the engine (Lin et al., 2024a) — while leaving the model weights untouched; DEP addresses the orthogonal memory axis, deciding *which experts* must be resident for each agent's turn, and could sit inside such engines.

B Experimental details

Workflow discovery and structure. Per task we run the AFlow MCTS search (Zhang et al., 2025) with a Claude-Sonnet optimizer and the target backbone as executor, for up to 10 rounds, and select the best-scoring discovered workflow on the search validation split. The discovered workflows are genuinely heterogeneous multi-agent programs; e.g. the MATH workflow runs five parallel solvers, self-consistency ensembling, a programmatic cross-check that writes and executes code, and a verifier (8 model calls per problem). For SWE-bench we use a fixed four-role workflow (Localizer $\rightarrow$ Programmer $\rightarrow$ Reviewer $\rightarrow$ Debugger) in an oracle-assisted setting: gold affected-file paths and failing-test names are provided in the prompt, which is why absolute resolve rates are not comparable to end-to-end agent leaderboards.

Discourse-operator taxonomy. The operator taxonomy of Section 2 adapts the segmentation *methodology* of Lee et al. (2026) but not its dictionary: that dictionary is built around reflective markers, and our traces are non-thinking ("wait" appears in 0.9% of messages, "hmm" in none), so forced matching produced labels the text does not support. We instead rediscover the taxonomy from the corpus itself: sentence-opening pivots are mined from 8,125 messages, the 311 accepted pivots are embedded and clustered, and a silhouette sweep over the number of clusters selects the nine committal operators used in the paper. The dictionary is then frozen and applied unchanged to every pool; Figure 2(c) reports every role with at least 50 segments. The operator analysis runs on nine task pools, a superset of the six benchmarks evaluated in Section 4.

Contamination control. Evaluation pools are disjoint from (i) the AFlow search-validation problems, (ii) every problem used to collect predictor-training transcripts, and (iii) every problem used to calibrate any static baseline; disjointness is verified by text-overlap checks before first evaluation on each task.

Predictor training. Transcripts are collected by running each discovered workflow with the dense backbone on its search-validation split. The six-task corpus is balanced on usable examples rather than problems (workflow depth differs per task): 379 problems yielding 1,542 per-agent-call examples, $\sim$253–261 per task, over 10 workflow nodes. One predictor per retention is trained with the objective of Eq. 6 (α=0.5) for 8 epochs: AdamW, learning rate 10^{-4}, weight decay 0.01, gradient clipping 1.0, accumulation 4, head bottleneck 1024 with GELU, soft-top-K temperature annealed $0.7 \rightarrow 0.15$. Training takes $\sim$3 GPU-hours per retention level on two 48 GB GPUs; no component of the backbone is updated.

Table 6: Objective ablation (ρ=50%, n=50). Both terms of Eq. 6 are load-bearing.

Objective	MATH	HumanEval+	HotpotQA (F1)	Mean
Dense reference	82.0	92.0	72.2	82.1
KD + BCE(REAP) (ours)	76.0	**90.0**	**71.7**	**79.2**
LM + BCE(REAP)	76.0	84.0	68.4	76.1
LM + BCE(norm)	**82.0**	80.0	0.0	54.0
KD only (α=1)	30.0	—	—	—

Table 7: Hyperparameter sweep (screened on MATH, the only task with headroom; paired on identical problems). All three knobs are null; apparent winners did not survive paired replication on an independent subset.

Knob	Values	Outcome
α	0, .25, .5, .75	flat ($\pm$3); α=1 collapses (above)
learning rate	$3{\cdot}10^{-5}$, 10^{-4}, $3{\cdot}10^{-4}$	U-shaped $\pm$5 $\Rightarrow$ noise
final temperature T	.05, .15, .4	flat

Efficiency measurement. Accuracy runs apply masks at the router (all experts resident), so memory and latency are measured separately. Resident-memory savings use the real delta-loading runtime of Section 3.4 — per-expert weights in pinned host memory, a resident buffer of K slots per layer with a global-to-slot index map, and the router masked to the resident set — measuring actual peak allocator usage and host-to-device swap latency as a function of the number of experts changed. Wall-clock generation time and token counts come from sequential, contention-free runs over three random 50-problem subsets (mean $\pm$ std).

Hardware. 30B-model experiments ran on 2×48 GB GPUs (generation and training) and the 480B model on 3×B200; SWE-bench patches are scored with the official Docker harness on a separate host.

C Ablations: the training recipe is robust and un-tuned

All ablations use a three-task subset of the main setting (MATH, HumanEval+, HotpotQA; discovered workflows, ρ=50%, n=50 held-out problems); every cell shares identical hyperparameters, and the sweeps were run *after* the main configuration was fixed — so the numbers in Section 4 are not the product of a hyperparameter search.

Both loss terms are load-bearing. Over a 4×5 grid of distillation objectives $\times$ saliency criteria, no cell beat the shipped combination (Table 6). Removing the criterion term entirely (α=1, pure self-distillation) collapses MATH from 76.0 to 30.0, while replacing the KD term with a plain masked language-modeling loss both degrades the mean and can produce *degenerate* predictors — the LM+norm cell decodes a single repeated token on 47/50 HotpotQA problems. Between these extremes the mixing weight is uncritical: $\alpha \in [0, 0.75]$ varies MATH by $\pm$3 points with no trend.

The saliency criterion is already near-optimal. Among the criteria in the grid, REAP (Eq. 4) is the best target; it also approximates an *exact* gradient-based Taylor saliency better (top-K set agreement 0.854) than routing frequency (0.780), output norm (0.839), or two purpose-built forward-only Taylor surrogates (0.796–0.799) — at zero extra cost, since it is computed in the same forward pass as the transcripts.

Everything else is null or protective. The full sweep (Table 7) found no improvement; the two apparent single-subset winners (α=.25, lr $3{\cdot}10^{-5}$) reversed or split under paired replication. The same hyperparameter set is used at both retentions — the ρ=50% and ρ=25% checkpoints differ in exactly one field, K. Finally, the transcript length filters are protective rather than incidental: admitting very short responses (fewer than 16 response tokens) adds examples whose per-example saliency is averaged over too few tokens to be meaningful, and costs 21 F1 points on the task those examples come from.

Table 8: Trained-on structures (32 workflows): only these train the universal predictor.

Task	Workflow	Operator sequence
HotpotQA	debate_1 r2	Custom → Custom → Custom → Custom
	debate_1 r3	Custom → Custom → Custom → Custom → Custom
	debate_1 r5	AnsGen → AnsGen → AnsGen → ScEns → Custom
	debate_1 r6	AnsGen → AnsGen → Custom → Custom
	decompose_1 r3	Custom → AnsGen → Custom → ScEns
	decompose_1 r6	Custom → AnsGen → ScEns → Custom
	decompose_1 r7	Custom → AnsGen → Custom
	decompose_2 r3	Custom → AnsGen → Custom
	direct_1 r1	AnsGen
	direct_1 r3	AnsGen → ScEns
	direct_2 r3	AnsGen → ScEns
	direct_2 r5	AnsGen → ScEns → Custom
HumanEval	direct_1 r1	Custom
	direct_1 r2	Custom → Test
	direct_1 r4	Custom → Custom → Custom
	direct_1 r7	Custom → Custom → Custom → Test
	ensembletest_1 r1	Custom → Test → ScEns
	ensembletest_1 r2	Custom → Test → ScEns → Custom → Test
	ensembletest_1 r6	Custom → Test → ScEns
	review_1 r1	Custom → Custom → Test
	review_2 r2	Custom → Custom → Custom → ScEns → Test
	review_2 r6	Custom → Custom → Custom → Test
	selfcons_1 r1	Custom → ScEns
	selfcons_1 r3	Custom → ScEns → Custom
	selfcons_2 r5	Custom → ScEns → Custom → Custom → Test
MATH	codefirst_1 r1	Programmer → Custom
	direct_1 r1	Custom
	direct_1 r2	Custom → ScEns
	direct_1 r3	Custom
	direct_1 r4	Custom → ScEns → Custom
	direct_2 r3	Custom → Custom
	selfcons_1 r2	Custom → ScEns → Custom

Table 9: Level 0 held-out — **unseen structures** (held-out rounds of families the predictor *did* train on).

Task	Workflow	Score	Operator sequence
HotpotQA	debate_1 r4	0.719	Custom → Custom → Custom → Custom → Custom
	decompose_2 r4	0.833	Custom → AnsGen → AnsGen → ScEns → Custom
	direct_1 r4	0.453	Custom → AnsGen
HumanEval	review_1 r2	0.900	Custom → Custom → ScEns → Test
	selfcons_1 r6	0.960	Custom → Custom → ScEns
MATH	direct_2 r2	0.833	Custom → ScEns
	selfcons_1 r1	0.767	Custom → ScEns

D WORKFLOW ROSTERS FOR THE GENERALIZATION LADDER

Tables 8–11 list every workflow used in the generalization study of Section 4.3: the 32 training workflows, and the held-out sets at each rung. Scores are the workflows' own validation scores under the dense backbone. AnsGen abbreviates AnswerGenerate and ScEns the self-consistency ensemble operator.

E PER-TASK EFFICIENCY RESULTS

Table 12 gives the per-task breakdown behind the averaged efficiency numbers of Table 3, with seed std.

Table 10: Level 1 held-out — **unseen families** (whole families never seen in training: *genverify* / *gentestfix* / *solveverify*).

Task	Workflow	Score	Operator sequence
HotpotQA	genverify_1 r1	0.713	AnsGen → Custom → Custom
	genverify_2 r3	0.793	AnsGen → ScEns → Custom → Custom
	genverify_2 r4	0.790	AnsGen → Custom → Custom → Custom
	genverify_2 r5	0.713	AnsGen → AnsGen → AnsGen → ScEns → Custom → Custom
HumanEval	gentestfix_1 r1	0.920	Custom → Test → Custom → Test
	gentestfix_1 r2	0.940	Custom → Custom → Test → Custom → Test
	gentestfix_1 r3	0.940	Custom → Custom → ScEns → Test → Custom → Test
MATH	solveverify_1 r1	0.767	Custom → Programmer → Custom
	solveverify_1 r2	0.900	Custom → Programmer → Custom
	solveverify_1 r3	0.767	Custom → ScEns → Programmer → Custom

Table 11: Levels 2–3 held-out — **unseen tasks**: MBPP = unseen code benchmark, MMLU = unseen knowledge domain.

Level / Task	Workflow	Score	Operator sequence
L2 · MBPP (code)	MBPP_1 r7	0.567	CodeGen → Custom → Test → Custom
	MBPP_1 r9	0.700	CodeGen → Test → Custom
	MBPP_1 r14	0.700	CodeGen → ScEns → Test → Custom
L3 · MMLU (knowledge)	MMLU_1 r1	0.867	Custom
	MMLU_1 r2	0.567	AnsGen → Custom → Custom → ScEns
	MMLU_1 r3	0.867	AnsGen → Custom → Custom → Custom → ScEns

F PER-TASK RESULTS ON A DIFFERENT MOE ARCHITECTURE

Table 13 gives the per-task breakdown behind the averaged DeepSeek-V2.5 results of Table 5.

G TOKEN WASTE AND GENERATION COLLAPSE

Table 14 traces where the static masks' extra wall time comes from, on HotpotQA at ρ=50% (N=50): they emit far more tokens per problem, and the excess concentrates in *max-token explosions* — generations that collapse into repetition loops and run to the cap. Dense and DEP never explodes; each static mask does, on different inputs. Table 15 shows one example of each.

H LIMITATIONS

This section expands the summary in Section 5.

Training cost and its scope. DEP needs transcripts of the dense backbone running the training workflows, and one predictor per retention level — our ρ=50% and ρ=25% checkpoints differ in exactly one field, K. The cost is small (roughly 3 GPU-hours per level, Appendix B) and is paid once rather than per configuration, but it is a step static calibration does not require, and serving a new retention level means training a new predictor.

The training recipe is heuristic and the gap to dense is open. Pruned accuracy still trails the dense workflow at both retentions: we claim graceful degradation under compression, not free compression. We do not believe the current recipe is near the ceiling. Its objective mixes a per-example saliency target with a self-distillation term in a fixed proportion, and the mask keeps a uniform top-K in every layer; both are heuristic choices. The sweeps in Appendix C explore only a neighbourhood of that design — mixing weight, learning rate, annealing temperature and the choice of saliency criterion — and leave structurally different alternatives untouched, such as an objective that scores a whole candidate mask rather than experts individually, or a global budget allocated unevenly across layers in place of a fixed K per layer. We expect a meaningful part of the remaining gap to dense to be recoverable along those lines.

Table 12: Per-task efficiency at ρ=50%: mean ± std over 3 random 50-problem seeds, sequential contention-free runs.

task	policy	accuracy	tokens/prob	generation (s)
HotpotQA	dense	.71±.03	172±8	412±70
	DEP	**.75±.09**	181±8	**405±42**
	REAP	.36±.06	224±24	780±96
	freq	.46±.08	477±28	894±26
HumanEval+	dense	.94±.03	1156±72	2548±266
	DEP	.90±.03	1120±42	**2391±130**
	REAP	.79±.02	1272±92	3113±318
	freq	.00±.00	3352±39	4746±15
MATH	dense	.69±.18	4387±114	6757±298
	DEP	.64±.13	4316±247	7030±346
	REAP	.45±.17	4734±259	7516±178
	freq	.36±.04	6266±43	10730±179

Table 13: DeepSeek-V2.5-236B (59 MoE layers, 160 routed experts, top-6, plus 2 shared experts that are always active and never pruned). Values are % (n=50 problems per task; pass@1, HotpotQA token-F1, MATH/GSM8K exact match); Dense and Direct are unpruned references. **Bold** = best pruned method per task.

Method	MATH	GSM8K	HumanEval+	MBPP+	HotpotQA	*Mean*
Dense (workflow)	52.0	84.0	72.0	86.0	59.7	70.7
Direct (single call)	50.0	90.0	76.0	84.0	56.9	71.4
50% retention (keep 80/160)						
Freq.	33.0	71.0	17.0	5.0	**39.2**	33.0
REAP	32.0	68.0	38.0	56.0	38.5	46.5
DEP	**34.0**	**74.0**	**46.0**	**66.0**	38.7	**51.7**
25% retention (keep 40/160)						
Freq.	1.0	2.0	0.0	0.0	1.0	0.8
REAP	2.0	2.0	0.0	2.0	1.7	1.5
DEP	**4.0**	**4.0**	**2.0**	**12.0**	**12.5**	**6.9**

Fine-grained expert pools are assumed, and few architectures are tested. Masking requires $K \geq k$, so retention below k/E cannot be expressed at all, and just above it the router has little freedom left inside the kept set. Every backbone we study has $E \geq 128$ routed experts; on coarse-grained MoEs with few experts per layer this constraint binds quickly and there are correspondingly few distinct masks to choose among, so we expect the benefit of per-request specialization to shrink as expert granularity coarsens. More broadly, our architectural coverage is thin: the backbones we evaluate span two model families, so architecture independence here is supported by example rather than established, and coarse-grained designs in particular remain untested.

The workload must actually be heterogeneous. DEP's advantage comes from agents that need different experts. A workflow whose nodes are near-identical gives the predictor little to specialize over, and DEP should then behave much like a static mask fitted to that one configuration.

Scope of the evidence. Our multi-agent workloads are automatically discovered workflows, and our agentic-coding benchmark is run oracle-assisted; neither is a production deployment, whose prompts, tool use, and traffic mix could differ in ways we cannot anticipate here. The largest backbone we study is 480B, so frontier-scale behavior is extrapolation. Validating DEP on production multi-agent systems and at frontier scale is the most useful next test.

Measurement caveats. Several evaluations use small per-cell sample sizes — stated in Section 4.1 and in each table caption — so individual cells carry wide intervals and small per-cell differences should not be read as an ordering. MATH is subset-sensitive in our harness — dense accuracy varies substantially between random draws of the same pool — so MATH numbers are interpretable only against a dense reference measured on the same problems. The mask-overlap analysis of Section 4.4 is measured on hand-crafted agent teams rather than the discovered workflows of Table 1. The Δ-load column of Table 3 is an analytic worst case rather than a measured swap latency, and the

Table 14: Token statistics of the final-answer agent (HotpotQA, ρ=50%, N=50). Exploded = generations $\geq$100 tokens; ms/token reflects batch padding, not per-token compute.

Policy	Accuracy (F1)	Tokens/prob	ms/token	Median tok	Max tok	# exploded
dense	0.73	180	47.3	4	21	0
DEP	0.70	**188**	45.5	4	11	0
REAP	0.36	259	69.9	7	128	3
freq	0.33	510	38.9	6	128	4

Table 15: Generation collapse under static masks: the same input, all four policies. Top: REAP collapses; bottom: freq collapses.

Policy	Output	Tok	Score
Q: "The saint in 'A Time for Miracles' was born on what date?"			
dense	`August 28, 1774`	10	1.00
DEP	`August 28, 1774`	10	1.00
REAP	`1774.1755-5.1815.1755-5.1815...` *(repeats to max)*	128	0.00
freq	`**Answer:**` *(empty)*	3	0.00
Q: "Gail Matthius co-anchored Weekend Update with the actor who played the villain Nicholas Andre in what movie?"			
dense	`Dumb and Dumber`	5	1.00
DEP	`"Dumb and Dumber"`	6	1.00
REAP	`Dumb and Dumber (1988)`	11	0.86
freq	`"D" in "D" in "D" in ...` *(repeats to max)*	128	0.00

prefetching and agent-batching that would reduce it in a production engine are described but not benchmarked end to end.