SKIM: Pruning Large Language Model Agents via Selective Knowledge Informed Masking

Moonseok Choi
KAIST
ms.choi@kaist.ac.kr

Jongwon Jeong
University of Wisconsin–Madison
jjeong64@wisc.edu

Minki Kang
KAIST
minkikang@kaist.ac.kr

Giung Nam[†]
KAIST
giung@kaist.ac.kr

Juho Lee[†]
KAIST
juholee@kaist.ac.kr

Abstract

Large Language Model (LLM) agents are increasingly deployed under structured generation, where grammar constraints enforce output format at inference time. Whereas existing compression methods target unconstrained generation, we first identify a fundamental mismatch: under structured generation, grammar constraints reduce the model's output distribution to valid tokens, and the logits matter most at the small set of positions that decide the agent's action. Motivated by this finding, we propose SKIM (Selective Knowledge-Informed Masking), the first pruning and distillation framework for LLM agents under structured generation, with both stages guided by token role. SKIM partitions token positions into context, decision, and format categories, and this partition drives two complementary axes: (1) category-aware pruning saliency scores that preserve parameters most relevant for decision positions; and (2) an on-policy distillation objective that aligns student and teacher on valid tokens, paired with category-aware feature matching. We evaluate SKIM on tool-calling benchmarks and within multi-step agent loops, across multiple model scales and sparsity levels. SKIM consistently improves accuracy over baselines, with its superiority becoming most striking in high-sparsity regimes where conventional methods struggle, demonstrating the potential of category-aware compression for deployment-time grammar-constrained agents. The project page is available at link.

1 Introduction

Large Language Model (LLM) agents are widely served under structured generation, where outputs must conform to a predefined format and tokens that would violate it are masked before sampling at every decoding step [40, 8]. This is now standard practice in production serving stacks [8, 47]. Yet the pruning and distillation methods used to compress LLMs are designed under the assumption of unconstrained generation [9, 38, 13, 28], and recent work confirms that off-the-shelf compression degrades sharply on tool-calling tasks [7].

The mismatch is *structural*, not incidental. Under structured generation, the model's effective distribution at each step is its raw softmax restricted to valid tokens and renormalized [40, 12]. The objectives

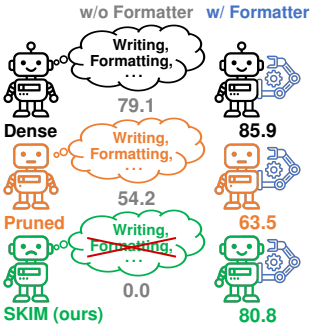


Figure 1: Target behavior.

[†]Co-corresponding authors.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

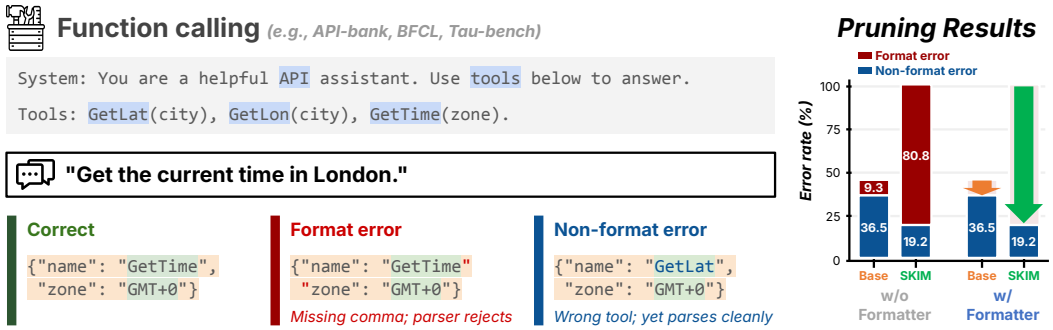


Figure 2: Token categories and error decomposition under structured generation. **Left:** Example function call with tokens highlighted by category (*Context*, *Decision*, *Format*) and three response types distinguishing format errors (eliminated by structured decoding) from non-format errors (which require the model itself to be correct). **Right:** Pruned-then-distilled model performance on Qwen3.5-2B [33] and BFCL-v3 [31] at 25% sparsity. SKIM fully exploits the grammar formatter: without it, SKIM looks broken because format errors dominate, but with it, SKIM easily beats Base (Wanda-sp). Format errors are essentially free under structured decoding, so reducing non-format errors at decision positions is what matters.

used in prior compression work, KL divergence to a teacher [13, 28] and layer-wise activation reconstruction [9, 38], are all computed against the unconstrained output. They preserve parameters that contribute to outputs the runtime masks, wasting capacity on behavior it already enforces.

Our key insight is that *not all output tokens play the same role under structured generation.* Hinging on this insight, we partition the agent's output into three roles: *(1) context* positions hold the input the model reads, *(2) decision* positions are the few output points where the format admits multiple valid options such as the function name or enum-valued arguments, and *(3) format* positions are the syntactic scaffolding the constraint mechanism enforces [8, 12]. The constraint mechanism enforces format validity for free at deployment [40, 8], so capacity allocated to format positions is wasted on behavior the runtime supplies regardless. Figure 2 (right) shows that even with standard distillation for healing [28, 37], the error of pruned agent models is dominated by non-format error, indicating that compression should preserve capacity at decision rather than format positions.

We propose **SKIM (Selective Knowledge-Informed Masking)**, the first pruning and distillation framework designed for LLM agents deployed under structured generation. The pruning side extends layer-wise reconstruction with per-category activation coefficients that down-weight format positions in the saliency score [38, 2]. The distillation side combines a forward-KL objective between the teacher's and student's constrained distributions on the student's own on-policy rollouts, paired with hidden-state matching at context and decision positions [28]. Both stages share the same token-role partition, aligning the preserved capacity with the deployment regime end-to-end.

We evaluate SKIM on tool-calling benchmarks, BFCL-v3 [31] and API-Bank L1 [19], and within ReAct-style agent loops [44] on HotpotQA [43]. Across model scales from 0.8B to 9B and sparsity levels, SKIM generally outperforms strong pruning and distillation baselines [38, 2, 22], with the largest margins at high sparsity where existing methods collapse but SKIM continues to recover most of the dense model's structured-decoding accuracy.

Our contributions are:

- To the best of our knowledge, we are the first to identify the mismatch between standard pruning and distillation objectives and constrained-decoding deployment, and we introduce a token-wise categorization of agent outputs into context, decision, and format positions, which grounds the design of our method (§ 3).
- We propose SKIM, a pruning and distillation framework that uses this categorization in both stages: a category-aware saliency that down-weights format positions in layer-wise reconstruction, and an on-policy distillation objective over the teacher's and student's constrained distributions paired with category-aware hidden-state matching (§ 4).

- We evaluate SKIM on tool-calling benchmarks and within multi-step agent loops across model scales from 0.8B to 9B, showing that SKIM preserves reasoning and tool-use capability with gains widening at higher sparsities (§ 5).

2 Related Work

Tool-calling LLM agents. Large language models are increasingly deployed as autonomous agents interacting with external APIs [44, 36, 30, 32, 46, 19]. As these agents transition to production-scale deployments, they face intertwined bottlenecks in efficiency and reliability. While prior work improved robustness via tool embeddings [15] or function masking [21], reliable deployment fundamentally requires syntactically flawless API requests (e.g., strict JSON). Because even minor structural deviations cause cascading execution failures, production pipelines overwhelmingly rely on mechanisms that explicitly enforce the desired output structure.

Structured generation. Structured generation enforces a grammar at inference time, guaranteeing structurally valid outputs. Early constrained-decoding algorithms [6, 11] formalized this idea, and modern engines [4, 40, 18] realize it efficiently by masking invalid tokens against predefined schemas or finite-state machines, which we call a *formatter* in Figs. 1 and 2. Their objective is faithful, low-overhead grammar enforcement, and the LLM is treated as a black box that the engine wraps without modification. A grammar, however, makes some token positions deterministic and others not - at deterministic positions the constraint uniquely fixes the next token, while at the rest the model's prediction selects the output. This position-level asymmetry is outside the scope of grammar-enforcement engines, but for compression methods designed for the structured-generation regime it is exactly the signal worth exploiting for compression.

LLM pruning. LLM pruning methods divide along granularity and saliency framework. Unstructured pruning methods score individual weights for sparse patterns [9, 38], while structured pruning methods remove whole architectural units for wall-clock speedups on dense hardware [25, 3, 26, 2, 22]. Cutting across both is *layer-wise reconstruction*, an influential saliency framework that scores units by how much zeroing them perturbs the layer's output on calibration activations [9, 38, 2, 22]. SKIM extends layer-wise reconstruction by partitioning calibration activations by the role each position plays under structured generation. We accordingly select Wanda [38], FLAP [2], and SlimGPT [22] as baselines, spanning unstructured and structured layer-wise reconstruction methods and providing the most direct comparison to SKIM.

LLM distillation. Knowledge distillation (KD) facilitates knowledge transfer from a teacher to a student model by matching output distributions or intermediate hidden states [16, 34]. While KD has long been a well-established paradigm in language modeling [17, 35, 39], recent research has shifted toward on-policy approaches, where students learn from their own generated trajectories to improve alignment in generative tasks [13, 1, 42]. Additionally, KD has been utilized to recover performance degradation following LLM pruning [28, 37]. However, to the best of our knowledge, no existing studies have explored KD specifically within the context of grammar-constrained decoding. For the detailed literature review, we refer readers to § A.

3 Preliminaries and Motivation

3.1 Preliminaries

Constrained language generation. Modern autoregressive language models define the conditional probability of a sequence $y \equiv y_{1:T} = (y_1, \ldots, y_T)$ given an initial prompt x as:

$$p(y_{1:T} \mid x) = \prod_{t=1}^{T} p(y_t \mid x, y_{<t}), \tag{1}$$

where x and y are sequences composed of tokens from a vocabulary $\mathcal{V}$, and generation proceeds from left to right. To incorporate specific requirements into the generation process (e.g., grammar constraints in formal language), we assume a potential function ϕ that assigns scores to sequences based on desired attributes. To satisfy constraints during generation, a widely adopted technique is

locally constrained decoding, which steers the generation process at each time step t by drawing tokens from a re-weighted density:

$$p_\phi(y_t \mid x, y_{<t}) \propto p(y_t \mid x, y_{<t}) \frac{\phi(x, y_{1:t})}{\phi(x, y_{<t})}. \tag{2}$$

In this work, we focus on *binary hard constraints* where $\phi \in \{0, 1\}$. We assume ϕ is monotonic such that if a prefix violates the constraints, any of its extensions will also be invalid, i.e., $\phi(x, y_{<t}) = 0$ implies $\phi(x, y_{1:t}) = 0$ for all $y_t \in \mathcal{V}$. Under this assumption, the ratio $\phi(x, y_{1:t})/\phi(x, y_{<t})$ is safely defined as zero whenever the denominator is zero. Such a condition is typically satisfied in *structured generation* tasks, where syntactic or formal constraints can be verified incrementally.

Remark. In practice, structured generation libraries [40, 8] enforce these constraints by defining a valid token set $\mathcal{V}_t \equiv \mathcal{V}(x, y_{<t}) = \{y_t \in \mathcal{V} \mid \phi(x, y_{1:t}) = 1\}$ at each time step t. Tokens not in $\mathcal{V}_t$ are masked by setting their logits to $-\infty$, restricting the sampling space of $p(\cdot \mid x, y_{<t})$ to $\mathcal{V}_t$.

Token position categories. For a sequence (x, y) comprising prompt x and output y, we partition token positions into three functional categories, *context* (c), *decision* (d), and *format* (f), as illustrated in Fig. 2. *Context positions* $P_c(x)$ are prompt-side positions, specifically the tokens of tool names, descriptions, and user queries. In contrast, *decision positions* $P_d(y) = \{t \in [T] : |\mathcal{V}_t| > 1\}$ are output positions where the schema leaves more than one valid continuation, requiring the model to choose among alternatives, whereas *format positions* $P_f(y) = \{t \in [T] : |\mathcal{V}_t| = 1\}$ are output positions where the schema admits a unique continuation, rendering the conditional distribution $p_\phi(\cdot \mid x, y_{<t})$ a point mass independent of the model's logits.

3.2 Motivation: format-position capacity is recoverable under structured decoding

Let (x, y^*) be a prompt and its corresponding reference response pair, where $\phi(x, y^*) = 1$. The cross-entropy of the ϕ-constrained neural network model p_ϕ with respect to this pair is given by:

$$\mathcal{L}_{\text{CE}}(p_\phi; x, y^*) = -\sum_{t=1}^{T} \log p_\phi(y_t^* \mid x, y_{<t}^*), \tag{3}$$

which is the quantity that must be preserved across both pruned and distilled models. Notably, for any format position $t \in P_f(y^*)$, the prediction is deterministic, such that $p_\phi(y_t \mid x, y_{<t}^*) = [y_t = y_t^*]$ for all $y_t \in \mathcal{V}$. Consequently, both the gradient and the Hessian of the loss with respect to any internal weight matrix W evaluate to zero at these specific positions:

$$\forall t \in P_f(y^*) : \nabla_W[-\log p_\phi(y_t^* \mid x, y_{<t}^*)] = 0 \;\text{ and }\; \nabla_W^2[-\log p_\phi(y_t^* \mid x, y_{<t}^*)] = 0. \tag{4}$$

While this property is absent in the layer-wise reconstruction loss typically used as a proxy in standard LLM pruning [9, 38], it reveals a fundamental asymmetry across token positions when evaluated under the structured cross-entropy loss in Eq. 3. Hinging on this, we investigate whether such positional asymmetry can be leveraged even within standard LLM pruning scenarios.

Empirical results. In addition to Fig. 2 (right), Fig. 5 shows the error decomposition aggregated over BFCL and API-Bank, two Qwen3.5 model sizes (0.8B and 2B), and three pruning baselines (Wanda [38], FLAP [2], SlimGPT [22]). Each baseline applies its own pruning mask and then heals the pruned model via knowledge distillation on xLAM-function-calling data [23] under a matched compute budget; structured generation at inference uses XGrammar [8]. Total error grows with sparsity, but non-format error dominates at every level: format errors are largely absorbed by the constraint mechanism, while non-format errors (wrong-tool-or-argument failures that determine the agent's action) remain the principal source of error. In § C.1, we further corroborate this pattern through a direct ablation that zeroes units specialized for each token category: removing format-specialized capacity barely changes accuracy, while removing context- or decision-specialized capacity collapses it (see Fig. 6). Together, these results suggest substantial room for improvement at decision-relevant positions, motivating SKIM's category-aware design.

Evaluation protocols. Throughout the paper, we adopt two metrics based on full-match accuracy against reference tool-call sequences for evaluation: (1) *structured accuracy* (i.e., $y \sim p_\phi$), which reflects the deployment setting under schema-constrained decoding and measures practical performance; (2) *free-generation accuracy* (i.e., $y \sim p$), a diagnostic under unconstrained decoding that isolates the model's intrinsic capabilities from the gains provided by the constraint mechanism ϕ.

Remark. A narrow gap between these metrics suggests that the model has natively dedicated its capacity to satisfying the constraints. Conversely, a large gap suggests it relies on ϕ to enforce the format, decoupling its intrinsic reasoning from the required structure.

4 SKIM

4.1 Layer-wise pruning with position-awareness

Reconstruction loss. Let $X \in \mathbb{R}^{T\times M}$ be the input activations and $W \in \mathbb{R}^{M\times N}$ be the original dense weight matrix in a neural network layer. The standard *layer-wise reconstruction loss* is:

$$\mathcal{L}(\hat{W}; W, X) = \left\| XW - X\hat{W} \right\|_F^2, \tag{5}$$

which measures how well the pruned weights $\hat{W}$ preserve the layer outputs. If we view the loss as a function of $\mathrm{vec}(\hat{W})$, its full Hessian is given by $\nabla^2\mathcal{L} = I_N \otimes H$ where $H := 2X^\top X$. Under the diagonal approximation $H \approx \mathrm{diag}(H)$, we ignore cross-feature interactions and obtain:

$$\mathcal{L}(\hat{W}; W, X) \approx \textstyle\sum_{i=1}^{M} \mathcal{L}_i, \quad \text{where } \mathcal{L}_i = \tfrac{1}{2} H_{ii} \left\| W_{i,:} - \hat{W}_{i,:} \right\|_2^2 \text{ and } H_{ii} = 2\textstyle\sum_{t=1}^{T} X_{ti}^2. \tag{6}$$

Gaussian interpretation. From a probabilistic perspective, Eq. 6 admits a Gaussian interpretation; up to an additive constant, $\mathcal{L}_i$ is the negative log-likelihood of the isotropic Gaussian model:

$$W_{i,:} \mid \hat{W}_{i,:}, X_{:,i} \sim \mathcal{N}(\hat{W}_{i,:}, \sigma_i^2 I_N), \tag{7}$$

with precision $1/\sigma_i^2 = H_{ii}$. Indeed, $\mathcal{L}_i$ corresponds to the quadratic term of the log-likelihood:

$$\log p(W_{i,:} \mid \hat{W}_{i,:}, X_{:,i}) = -\tfrac{N}{2}\log(2\pi\sigma_i^2) - \tfrac{1}{2\sigma_i^2}\left\| W_{i,:} - \hat{W}_{i,:} \right\|_2^2, \tag{8}$$

which implies that a larger sum of squared activations assigns a smaller variance, making the likelihood more sensitive to perturbations in the corresponding row of the weight matrix. The saliency of the i^{th} neuron is then evaluated as the decrease in the log-likelihood when it is zeroed ($\hat{W}_{i,:} = 0$), which exactly corresponds to the square of the heuristic pruning metric proposed in [38]:

$$S_i = \tfrac{1}{2} H_{ii} \left\| W_{i,:} \right\|_2^2 = \left(\textstyle\sum_{t=1}^{T} X_{ti}^2 \right) \left\| W_{i,:} \right\|_2^2. \tag{9}$$

Position-aware saliency score. However, Eq. 9 weights all token positions uniformly through the sum over t. Motivated by the observation that token contributions along the sequence dimension are not uniform, we introduce a context-weighted heteroscedastic surrogate precision:

$$\tilde{H}_{ii}^{\text{pos}} := 2 \cdot \left(\alpha_c \textstyle\sum_{t\in P_c} X_{ti}^2 + \alpha_d \sum_{t\in P_d} X_{ti}^2 + \alpha_f \sum_{t\in P_f} X_{ti}^2 \right), \tag{10}$$

where P_c, P_d, P_f form exclusive positional groups corresponding to context, decision, and format token roles, respectively, and $\alpha_c, \alpha_d, \alpha_f \geq 0$. Using $1/\sigma_i^2 := \tilde{H}_{ii}^{\text{pos}}$ in the saliency formulation yields the corresponding position-aware saliency score:

$$S_i^{\text{pos}} = \left(\alpha_c \textstyle\sum_{t\in P_c} X_{ti}^2 + \alpha_d \sum_{t\in P_d} X_{ti}^2 + \alpha_f \sum_{t\in P_f} X_{ti}^2 \right) \cdot \left\| W_{i,:} \right\|_2^2. \tag{11}$$

Remark. Notably, Eq. 11 can be interpreted as being derived from a weighted layer-wise reconstruction loss, i.e., $\mathcal{L}(\hat{W}; W, X) := \|A^{1/2}(XW - X\hat{W})\|_F^2$, where A is a diagonal scaling matrix with entries $A_{tt} = \alpha_c[t \in P_c] + \alpha_d[t \in P_d] + \alpha_f[t \in P_f]$. Nevertheless, a key advantage of our formulation lies in the modularity of the surrogate precision. By decoupling position-awareness from the loss objective and embedding it directly into the precision term, our framework is not restricted to any specific gating mechanism. Instead, it establishes a generalized design space: one can readily define diverse functional forms for $\tilde{H}_{ii}^{\text{pos}}$ to incorporate different positional inductive biases, enabling the systematic derivation of a wide family of position-aware saliency scores without altering the underlying standard layer-wise reconstruction loss in Eq. 5. In what follows, we experimentally compare various surrogate precision designs through an ablation study in § B.1.

4.2 SKIM Distillation

Distillation loss. Let p be the original dense model and q be the pruned sparse model. In practice, to recover the performance degradation from pruning, knowledge distillation [16] is often employed by minimizing a forward KL divergence between the two models on a dataset of (x, y) pairs [28]:

$$D_{\mathrm{KL}}(p \parallel q)(y \mid x) = \sum_{t=1}^{T} D_{\mathrm{KL}}(p(\cdot \mid x, y_{<t}) \parallel q(\cdot \mid x, y_{<t})). \tag{12}$$

In a constrained generation setup, however, the distillation loss should be reformulated to minimize the divergence between the ϕ-constrained distributions, p_ϕ and q_ϕ:

$$D_{\mathrm{KL}}(p_\phi \parallel q_\phi)(y \mid x) = \sum_{t=1}^{T} D_{\mathrm{KL}}(p_\phi(\cdot \mid x, y_{<t}) \parallel q_\phi(\cdot \mid x, y_{<t})), \tag{13}$$

where the t^{th} term in the summation decomposes as

$$\tfrac{1}{Z_p} \sum_{y \in \mathcal{V}_t} p(y \mid x, y_{<t}) \left[\log \tfrac{p(y|x,y_{<t})}{q(y|x,y_{<t})} - \log Z_p + \log Z_q \right], \tag{14}$$

where Z_p and Z_q are the normalization constants for p_ϕ and q_ϕ, required to yield valid probability distributions from the unnormalized formulations in Eq. 2. Notably, the logit masking technique for structured generation remains applicable here, allowing for the direct computation of the KL divergence over the valid vocabulary $\mathcal{V}_t$ without explicitly evaluating the normalization constants:

$$D_{\mathrm{KL}}(p_\phi \parallel q_\phi)(y \mid x) = -\sum_{t=1}^{T} \sum_{y_t \in \mathcal{V}_t} p_\phi(y_t \mid x, y_{<t}) \log q_\phi(y_t \mid x, y_{<t}) + \text{const}. \tag{15}$$

Remark. The decomposition in Eq. 14 clarifies why the constrained form is appropriate here. While the gradient of the first term in the bracket with respect to the k^{th} logit of q is non-zero for all k, the cancellation of $\log Z$ terms yields zero gradients for the logits corresponding to $\mathcal{V} \backslash \mathcal{V}_t$. Consequently, the loss only shapes the student's distribution over $\mathcal{V}_t$, which is the only factor that matters during inference under structured decoding. Such surgical specificity is crucial for tool-calling LLMs where shared parameters govern both constrained and unconstrained generation. Unlike vanilla distillation in Eq. 12, which unnecessarily suppresses off-support tokens, our approach in Eq. 15 avoids such updates, preventing performance leakage into unconstrained behaviors like free-form chat.

While Eqs. 12 and 15 minimize forward KL divergence to align token-level distributions, they leave open various design choices. Key among these are the source of y (e.g., original dataset, teacher-, or student-generated) and the specific divergence measure (e.g., forward or reverse KL) employed for matching [1]. As $\mathcal{V}_t$ and the later decision positions depend on the student's earlier decisions, we adopt, after an off-policy warm-up, an *on-policy forward KL* on student-generated responses y:

$$\mathcal{L}_{\mathrm{KD}}(q; p) = \mathbb{E}_{x \in \mathcal{D}} \mathbb{E}_{y \sim \mathrm{sg}(q_\phi(\cdot|x))} \left[D_{\mathrm{KL}}(p_\phi \parallel q_\phi)(y \mid x) \right]. \tag{16}$$

We further adopt feature distillation [34, 24, 28], but refine its scope to focus primarily on *context* and *decision* token positions. Indeed, our empirical findings in § 5.6 indicate that down-weighting *format* positions improves results, aligning with our argument that the limited capacity of the pruned model should be strategically allocated. The final distillation loss for SKIM is formulated as:

$$\mathcal{L}_{\mathrm{SKIM}} = \mathcal{L}_{\mathrm{KD}} + \lambda_H \cdot \mathcal{L}_{\mathrm{hidden}}, \quad \text{where } \mathcal{L}_{\mathrm{hidden}} = \sum_{\ell \in L} \sum_t \alpha_{\mathrm{cat}(t)} \cdot \left\| h_t^{(p,\ell)} - h_t^{(q,\ell)} \right\|_2^2, \tag{17}$$

where $h_t^{(p,\ell)}$ and $h_t^{(q,\ell)}$ are the teacher and student hidden states, respectively, at position t following transformer block ℓ. For further empirical details, we refer readers to § B.2.

5 Experiments

5.1 Experimental setup

Models and datasets. We evaluate SKIM across four scales of the Qwen3.5 series [33]: Qwen3.5-0.8B/2B/4B/9B. Dense models serve as reference. xLAM-function-calling-60k [23] provides activation statistics for pruning calibration and training data for distillation. We evaluate on two tool-calling benchmarks: BFCL-v3 [31] and API-Bank L1 [19], reporting full-match accuracy against reference tool calls. For the agentic pipeline evaluation, we use HotpotQA [43] dataset under ReAct-style loop [44], reporting F1 metric. Numbers are reported under structured decoding (the deployment setting) and free generation (see § 3.2). All decoding is greedy ($\tau = 0$).

Table 1: Post-KD full-match accuracy of Qwen3.5 series across varying sparsity ratios with thinking mode enabled. All reported numbers are measured under the same distillation budget. Each cell reports **structured-generation** / free-generation accuracy. SKIM achieves the best structured-generation accuracy in most settings, with the gap to baselines widening at higher sparsity.

		BFCL-v3			API-Bank L1		
	Method / Sparsity	10%	25%	50%	10%	25%	50%
0.8B	*Dense*		*78.0* / *72.6*			*57.1* / *47.6*	
	Wanda-sp	72.9 / 60.2	53.6 / 30.4	7.5 / 0.0	51.9 / 50.1	35.6 / 29.3	4.0 / 0.0
	FLAP	74.9 / 47.0	63.9 / 57.0	42.6 / 34.0	51.9 / 46.8	45.8 / 43.5	33.5 / 21.0
	SlimGPT	74.5 / 57.0	59.4 / 48.1	35.9 / 13.4	**55.4** / 49.9	44.1 / 39.6	28.3 / 9.5
	SKIM (ours)	**76.4** / 0.0	**71.1** / 0.0	**53.4** / 1.5	52.1 / 0.0	**46.9** / 0.3	**39.3** / 0.3
2B	*Dense*		*85.9* / *79.1*			*61.4* / *44.6*	
	Wanda-sp	78.9 / 54.8	63.5 / 54.2	24.9 / 12.9	**57.6** / 56.9	47.9 / 35.6	24.3 / 4.8
	FLAP	**83.8** / 82.1	75.9 / 73.0	56.2 / 46.5	56.6 / 53.1	52.6 / 51.1	38.1 / 35.3
	SlimGPT	82.0 / 74.4	73.6 / 49.1	49.0 / 38.1	56.6 / 51.9	49.9 / 47.1	34.1 / 26.8
	SKIM (ours)	82.9 / 7.0	**80.8** / 0.0	**64.5** / 0.4	**57.6** / 2.8	**54.5** / 3.0	**48.9** / 0.5
4B	*Dense*		*88.5* / *81.2*			*70.2* / *67.9*	
	Wanda-sp	85.9 / 82.9	70.5 / 43.9	23.4 / 8.2	58.9 / 49.9	58.1 / 53.6	15.0 / 0.8
	FLAP	85.8 / 84.1	79.9 / 76.2	57.6 / 48.5	62.4 / 58.9	60.4 / 58.9	42.1 / 37.3
	SlimGPT	86.0 / 83.1	82.4 / 80.1	61.3 / 37.9	**66.4** / 64.7	59.6 / 49.6	41.9 / 13.3
	SKIM (ours)	**86.1** / 20.6	**85.0** / 44.6	**75.1** / 0.0	63.2 / 2.0	**62.2** / 6.8	**51.4** / 0.0
9B	*Dense*		*92.0* / *86.1*			*75.9* / *71.4*	
	Wanda-sp	87.5 / 82.6	78.4 / 52.5	24.1 / 5.4	67.4 / 13.8	61.2 / 20.1	21.6 / 0.0
	FLAP	87.8 / 85.5	81.9 / 75.6	48.1 / 31.1	**70.7** / 44.0	61.9 / 23.7	39.1 / 7.3
	SlimGPT	88.6 / 87.0	**86.4** / 82.8	59.1 / 62.5	67.7 / 33.3	58.9 / 47.6	48.1 / 43.1
	SKIM (ours)	**90.2** / 81.0	85.5 / 71.0	**77.6** / 1.3	65.6 / 44.4	**66.9** / 53.1	**58.4** / 0.5

Sparsity levels. We report sparsity at 10%, 25%, and 50%, applied as structured sparsity over the prunable-unit set (MLP output channels, attention KV groups, and linear-attention groups). Following the convention in prior structured-pruning work [2, 22], these rates are computed relative to the prunable parameters only; modules outside this set (token embeddings and the LM head) are not pruned. The corresponding total parameter reductions are roughly 6–7%, 15–18%, and 30–36% across the four model scales – for instance, 6.77%, 17.07%, and 34.14% on Qwen3.5-2B (see § C.3 for the full parameter and memory breakdown).

Baselines. Following § 2, we compare against three layer-wise reconstruction baselines that share SKIM's framework and provide the most direct comparison: Wanda-sp (the structured variant of Wanda [38]), FLAP [2], and SlimGPT [22]. Each baseline is pruned with its own saliency criterion and then healed with standard output-matching knowledge distillation on the fixed xLAM-FC subset, under matched training compute. As Qwen3.5 series are not tool-specialists, we fine-tune dense models as well for a fair comparison across methods.

SKIM settings. We use category coefficients $(\alpha_c, \alpha_d, \alpha_f) = (1.0, 1.0, 0.2)$, fixed across models and sparsity levels, and unnormalized feature-distillation weight $\lambda_H = 0.5$. We label context token categories using GPT-4 [29]. Distillation is run for 2 epochs at a learning rate of 1×10^{-5} on a 4,096-sample subset of xLAM-FC, with maximum sequence length 1,300 and maximum generation length 300. Detailed hyperparameters and implementation specifics are deferred to § B.

5.2 Main results: tool-calling benchmarks

Quantitative analysis. SKIM generally achieves the best structured-decoding accuracy in Table 1, with the gap to baselines widening as sparsity increases. At 50% sparsity, SKIM retains 68–85% of dense accuracy, and smaller SKIM-pruned models can match or exceed baseline pruning applied at the next-larger scale. Free-generation accuracy drops sharply, quite frequently to near zero, as the constrained-distribution distillation objective (Eq. 15) updates the student's logits only on the valid vocabulary at each step; this is by design for the deployment regime (§4.2). The 9B models

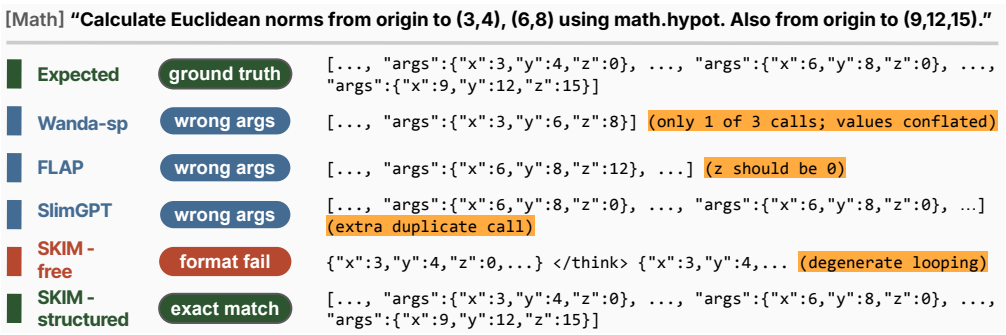


Figure 3: BFCL-v3 math example on baseline methods and SKIM. SKIM-free's format errors are intentional: the freed capacity lets SKIM-structured produce the correct call. Baselines fail with wrong arguments or wrong tools, errors grammar cannot fix. See § D for more examples.

Table 2: Post-KD full-match accuracy of Hammer2.1-3B at various sparsity ratios with thinking mode enabled. Each cell reports **structured-generation** / free-generation accuracy. On the tool-specialist model, which is already fine-tuned for function calling, SKIM still achieves the best structured-generation accuracy at 50% sparsity on both benchmarks.

		BFCL-v3			API-Bank L1		
	Method / Sparsity	10%	25%	50%	10%	25%	50%
	Dense	*89.4* / *79.9*			*76.2* / *75.9*		
3B	Wanda-sp	88.6 / 87.4	83.8 / 81.0	72.2 / 65.5	**76.9** / 77.2	73.4 / 73.9	55.4 / 54.1
	FLAP	**89.9** / 88.6	88.0 / 86.1	73.0 / 68.6	75.9 / 74.9	**73.9** / 70.7	47.3 / 30.1
	SlimGPT	89.4 / 87.6	85.1 / 82.6	46.4 / 36.1	75.9 / 77.4	72.2 / 71.4	39.1 / 30.8
	SKIM (ours)	88.8 / 83.0	**88.1** / 80.4	**82.8** / 69.9	76.2 / 70.9	73.4 / 66.9	**67.7** / 14.0

at low sparsity are an exception, retaining substantial free-generation accuracy; this is consistent with the larger model having enough remaining capacity to keep free-form behavior alive despite the constrained objective. We also run SKIM alone on Qwen3.5-27B, without baselines at this scale; § C.5 reports its structured accuracy on BFCL-v3 over three seeds.

Qualitative analysis. Figure 3 shows a single BFCL-v3 example (math) comparing baselines, SKIM-free, and SKIM-structured. The design trade-off behind SKIM is visible: it under-allocates capacity at format positions, so SKIM-free produces malformed JSON, but that same allocation is what lets SKIM-structured pick the correct function and argument once the grammar formatter supplies the surrounding syntax. Baselines instead distribute capacity uniformly and fail at decision positions, producing well-formed JSON with the wrong tool or argument value. For the detailed prompt and further examples, we refer readers to § D.

5.3 Cross-model generalization

Table 2 repeats the evaluation on Hammer2.1-3B [21], a tool-calling model built on Qwen2.5-Coder. The Qwen3.5 pattern reproduces, with the gap to the strongest baseline widening as sparsity grows: SKIM is competitive at 10% and 25% sparsity, and dominates at 50%. The cross-backbone agreement supports the claim that SKIM's gains follow from the structure of constrained generation rather than from properties of any specific model architecture. Hammer also retains substantial free-generation accuracy after SKIM, in contrast to the near-zero collapse on Qwen3.5, because Hammer is already fine-tuned on function-calling data, which leaves its tool-calling priors largely intact. See § C.2 for a further analysis of the Hammer2.1-3B results.

5.4 Agentic reasoning benchmark

Table 3 evaluates Qwen3.5-2B inside a multi-step agent loop on HotpotQA [43], where the agent issues retrieval calls and reasons over their outputs. SKIM achieves the best performance at every sparsity level, consistent with our design: it explicitly retains reasoning capacity in exchange for for-

Table 3: Post-KD F1 on HotpotQA at various sparsity ratios with thinking mode enabled. SKIM achieves the best F1 at all sparsity ratios.

Method / Sparsity	10%	25%	50%
Dense		43.6	
Wanda-sp	19.2	14.0	11.1
FLAP	21.6	14.5	13.3
SlimGPT	20.8	19.5	10.8
SKIM (ours)	**31.7**	**20.2**	**14.9**

Table 4: Spider (component F1) and GSM8K (exact match) on Qwen3.5-2B at 10% sparsity. The baseline prunes with position-uniform saliency and heals with vanilla KD. Each cell reports **structured** / free performance.

Method	Spider	GSM8K
Dense	*63.5* / *16.9*	*29.5* / *14.5*
Baseline	53.1 / 16.8	21.0 / 22.5
SKIM (ours)	**56.4** / 7.8	**24.0** / 21.5

mat capacity, and reasoning is what multi-hop agentic loops most require. Pruned models still lose substantial performance relative to the dense baseline; this drop reflects a distillation-data mismatch, since we heal on single-turn traces rather than multi-step agent trajectories, and trajectory-level healing closes most of the gap (§ C.7). SKIM nevertheless remains the strongest pruned network at every sparsity ratio, in line with its lead in most settings of Table 1.

5.5 Beyond tool calling

SKIM can be applied to any task decoded under a grammar. Table 4 evaluates two further tasks on Qwen3.5-2B at 10% sparsity: text-to-SQL on Spider [45], where the SQL keywords and punctuation fixed by the grammar are format positions, and GSM8K [5] with each solution written as a JSON object of reasoning steps and a final answer, where the JSON syntax and field names are format positions. On both tasks SKIM improves the structured metric over the baseline; the free-generation drop is small on GSM8K, where the JSON template is a small part of each output.

5.6 Ablation studies

Category weights. Figure 4 sweeps α_f, which enters both pruning and hidden-state matching, on Qwen3.5-2B and API-Bank L1 at 25% sparsity, holding $\alpha_c = \alpha_d = 1.0$ fixed. Structured accuracy is non-monotonic in α_f. Setting $\alpha_f = 1.0$ weights the three categories equally, so format positions absorb capacity that should go to context and decision; setting $\alpha_f = 0$ discards the signal that format positions still carry for reconstruction. Structured accuracy peaks near $\alpha_f = 0.2$, matching the finding in § C.1 that format-specialized capacity is dispensable but not entirely useless. Free-generation accuracy moves in the opposite direction, rising as α_f increases over most of the sweep, confirming the design intuition that α_f acts as a structured-versus-free trade-off knob.

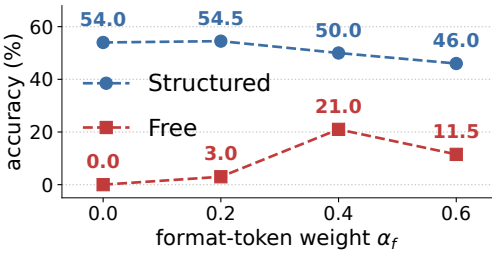


Figure 4: Format-token weight ablation on Qwen3.5-2B (25% sparsity) with API-Bank L1. Higher α_f trades structured accuracy for free-generation accuracy, as intended.

Distillation: loss components. Table 5 ablates the components of SKIM's distillation recipe on Qwen3.5-2B and BFCL-v3 data at 25% sparsity, and Table 6 varies the weight of the vanilla output-matching KD when it is added to the recipe. Removing the on-policy phase causes a substantial drop in structured accuracy because the student is then trained only on ground-truth prefixes, whereas at deployment it must continue from its own, possibly different, decisions. Removing the feature-distillation term causes a smaller drop, since the hidden-state target adds a coarser but still useful signal at context and decision positions. Adding the vanilla output-matching KD with coefficient $\beta = 0.5$ causes the largest single drop in structured accuracy but helps retain free-generation accuracy, and the sweep in Table 6 makes the structured-vs-free trade-off explicit: pure SKIM has

Table 5: Loss ablation on Qwen3.5-2B (25% sparsity) with BFCL-v3. Each variant degrades structured accuracy; the full SKIM is the best.

Variant	Structured	Free
SKIM (full)	**80.8**	0.0
(−) w/o on-policy.	78.1	0.0
(−) w/o feature distill.	79.1	0.5
(+) w/ vanilla KD.	77.9	46.4

Table 6: Accuracy on Qwen3.5-2B (25% sparsity, BFCL-v3) across vanilla-KD coefficients; all rows heal one SKIM-pruned model.

Objective	Structured	Free
Vanilla KD only	70.0	54.0
SKIM, $\beta = 0.50$	77.9	46.4
SKIM, $\beta = 0.25$	78.4	34.7
SKIM, $\beta = 0$	**80.8**	0.0

Table 7: Structured accuracy on BFCL-v3 with LLM (GPT-4), tool-span-rule, and full-context labels at 25% and 50% sparsity.

	Qwen3.5-2B		Qwen3.5-9B	
Context labels	25%	50%	25%	50%
LLM (default)	80.8	64.5	85.5	77.6
Tool-span rule	81.7	64.3	84.0	73.0
Full context	79.8	66.8	78.0	70.0

the highest structured accuracy and no free-generation accuracy, vanilla distillation alone the opposite, and intermediate coefficients give operating points between the two. Driving free-generation accuracy to zero is the right choice only for a specialist served behind a formatter; when one compressed model must also answer unconstrained requests, the coefficient β, like α_f in pruning, is the knob that sets how much free-generation accuracy is kept.

Context labeling without GPT-4. Table 7 replaces the GPT-4 context labels with the rule-based tool-span labeler of § B.1 and with labeling every prompt token as context; the rule matches GPT-4 labeling on Qwen3.5-2B and trails it by 1.5–4.6 points on Qwen3.5-9B, whereas labeling the full prompt as context degrades accuracy substantially on Qwen3.5-9B.

5.7 Efficiency

SKIM physically removes MLP channels and attention components, so the pruned checkpoint is a smaller dense model that runs without sparse kernels. Table 8 reports decoding throughput and prefill speed relative to the dense model at 50% nominal sparsity, measured on one 48 GB A6000 for the 2B and 9B models and on two for the 27B model. Width pruning keeps the sequential depth, so single-request speedups are modest; the gains grow with model size and batch size, reaching 1.59× throughput at batch size 256 for the 27B model, and § C.4 reports batch size one as well.

Table 8: Speedups of SKIM over the dense model at 50% nominal sparsity (decoding throughput at batch size 256 and prefill speed for 4,096 tokens).

Model	Decoding	Prefill
Qwen3.5-2B	1.29×	1.23×
Qwen3.5-9B	1.41×	1.37×
Qwen3.5-27B	1.59×	1.51×

6 Conclusion

We have presented SKIM, a pruning-and-distillation framework for LLM agents deployed under structured generation. SKIM operates from the observation that tokens in a tool-calling output do not all play the same role at deployment: a pruned model that preserves predictive fidelity uniformly across positions wastes capacity on outputs the constraint mechanism already guarantees. SKIM exploits a context/decision/format partition of token positions in both stages, with a category-aware pruning saliency derived from a heteroscedastic surrogate precision and a distillation objective that minimizes forward KL on the constrained student distribution with position-aware hidden-state matching. Across the Qwen3.5 series and a Hammer-2.1-3B backbone, SKIM generally achieves the best structured-decoding accuracy on BFCL-v3 and API-Bank, with the largest gains at 50% sparsity where existing methods struggle the most.

Future work and limitations. We discuss the limitations of our work and outline directions for future work. We evaluated SKIM on models up to 27B; validation at larger scales and on non-Qwen families remains future work, and the larger-compute regime may surface trends invisible at our current scale. Our distillation corpus consists of single-turn function-calling traces, so SKIM recovers multi-step agentic behavior only indirectly; distillation on multi-step agentic trajectories largely closes the HotpotQA gap. The pruning stage adopts a layer-wise reconstruction framework, leaving category-aware extensions to end-to-end pruning methods [25, 41], and to calibration-based quantization [10, 20] open. More broadly, SKIM instantiates a general principle that constrained inference changes which output positions the model controls and therefore which weights compression should preserve, with applications to code generation and other constrained-output settings.

Acknowledgements

This work was supported by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.RS-2019-II190075, Artificial Intelligence Graduate School Program(KAIST)). This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (RS-2022-NR070855). This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) with a grant funded by the Ministry of Science and ICT (MSIT) of the Republic of Korea in connection with the Global AI Frontier Lab International Collaborative Research Agreement (No. RS-2024-00469482 & RS-2024-00509279). We declare no competing interests.

References

[1] Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations (ICLR)*, 2024.

[2] Yongqi An, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. Fluctuation-based adaptive structured pruning for large language models. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2024.

[3] Saleh Ashkboos, Maximilian L. 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 (ICLR)*, 2024.

[4] Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Prompting is programming: A query language for large language models. *Proceedings of the ACM on Programming Languages*, 7 (PLDI):1946–1969, 2023.

[5] 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.

[6] Daniel Deutsch, Shyam Upadhyay, and Dan Roth. A general-purpose algorithm for constrained sequential inference. In *Conference on Computational Natural Language Learning (CoNLL)*, 2019.

[7] Peijie Dong, Zhenheng Tang, Xiang Liu, Lujun Li, Xiaowen Chu, and Bo Li. Can compressed LLMs truly act? An empirical evaluation of agentic capabilities in LLM compression. In *International Conference on Machine Learning (ICML)*, 2025.

[8] Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. XGrammar: Flexible and efficient structured generation engine for large language models. In *Machine Learning and Systems (MLSys)*, 2025.

[9] Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning (ICML)*, 2023.

[10] 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.

[11] Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured NLP tasks without finetuning. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

[12] Saibo Geng, Hudson Cooper, Michał Moskal, Samuel Jenkins, Julian Berman, Nathan Ranchin, Robert West, Eric Horvitz, and Harsha Nori. JSONSchemaBench: A rigorous benchmark of structured outputs for language models, 2025. URL https://arxiv.org/abs/2501.10868.

[13] Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: Knowledge distillation of large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

[14] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

[15] Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. ToolkenGPT: Augmenting frozen language models with massive tools via tool embeddings. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[16] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

[17] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP*, 2020.

[18] Alexander K. Lew, Tan Zhi-Xuan, Gabriel Grand, and Vikash K. Mansinghka. Sequential Monte Carlo steering of large language models using probabilistic programs, 2023. URL https://arxiv.org/abs/2306.03081.

[19] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

[20] 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 on-device LLM compression and acceleration. In *Machine Learning and Systems (MLSys)*, 2024.

[21] Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, Jun Wang, and Weinan Zhang. Hammer: Robust function-calling for on-device language models via function masking. In *International Conference on Learning Representations (ICLR)*, 2025.

[22] Gui Ling, Ziyang Wang, Yuliang Yan, and Qingwen Liu. SlimGPT: Layer-wise structured pruning for large language models. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[23] Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. APIGen: Automated PIpeline for generating verifiable and diverse function-calling datasets. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[24] Chengqiang Lu, Jianwei Zhang, Yunfei Chu, Zhengyu Chen, Jingren Zhou, Fei Wu, Haiqing Chen, and Hongxia Yang. Knowledge distillation of transformer-based language models revisited, 2022. URL https://arxiv.org/abs/2206.14366.

[25] Xinyin Ma, Gongfan Fang, and Xinchao Wang. LLM-Pruner: On the structural pruning of large language models. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

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

[27] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[28] Saurav Muralidharan, Sharath Turuvekere Sreenivas, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. Compact language models via pruning and knowledge distillation. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[29] OpenAI. GPT-4 technical report, 2023. URL https://arxiv.org/abs/2303.08774.

[30] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[31] Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The Berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *International Conference on Machine Learning (ICML)*, 2025.

[32] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations (ICLR)*, 2024.

[33] Qwen Team. Qwen3.5: Towards native multimodal agents, 2026. URL https://qwen.ai/blog?id=qwen3.5.

[34] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. FitNets: Hints for thin deep nets. In *International Conference on Learning Representations (ICLR)*, 2015.

[35] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter, 2019. URL https://arxiv.org/abs/1910.01108.

[36] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[37] Sharath Turuvekere Sreenivas, Saurav Muralidharan, Raviraj Joshi, Marcin Chochowski, Ameya Sunil Mahabaleshwarkar, Gerald Shen, Jiaqi Zeng, Zijia Chen, Yoshi Suhara, Shizhe Diao, Chenhan Yu, Wei-Chun Chen, Hayley Ross, Oluwatobi Olabiyi, Ashwath Aithal, Oleksii Kuchaiev, Daniel Korzekwa, Pavlo Molchanov, Mostofa Patwary, Mohammad Shoeybi, Jan Kautz, and Bryan Catanzaro. LLM pruning and distillation in practice: The Minitron approach, 2024. URL https://arxiv.org/abs/2408.11796.

[38] Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

[39] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for BERT model compression. In *Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019.

[40] Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models, 2023. URL https://arxiv.org/abs/2307.09702.

[41] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared LLaMA: Accelerating language model pre-training via structured pruning. In *International Conference on Learning Representations (ICLR)*, 2024.

[42] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[43] 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 *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[44] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[45] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[46] Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Zhiwei Liu, Yihao Feng, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. xLAM: A family of large action models to empower AI agent systems. In *Nations of the Americas Chapter of the Association for Computational Linguistics (NAACL)*, 2025.

[47] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

A Further Related Works

Tool-calling LLM agents. Large language models are widely deployed as autonomous agents interacting with external APIs [44, 36, 30, 32, 46, 19]. As these agents transition to production-scale deployments, they face intertwined bottlenecks in efficiency and reliability. While prior work improved robustness via tool embeddings [15] or function masking [21], reliable deployment fundamentally requires syntactically flawless API requests (e.g., strict JSON). Because even minor structural deviations cause cascading execution failures, production pipelines overwhelmingly rely on mechanisms that explicitly enforce the desired output structure.

Structured generation. Structured generation enforces a grammar at inference time, guaranteeing structurally valid outputs. Early constrained-decoding algorithms [6, 11] formalized this idea, and modern engines [4, 40, 18] realize it efficiently by masking invalid tokens against predefined schemas or finite-state machines. Their objective is faithful, low-overhead grammar enforcement, and the LLM is treated as a black box that the engine wraps without modification. A grammar, however, makes some token positions deterministic and others not - at deterministic positions the constraint uniquely fixes the next token, while at the rest the model's prediction selects the output. This position-level asymmetry is outside the scope of grammar-enforcement engines, but for compression methods designed for the structured-generation regime it is exactly the signal worth exploiting for compression under structured generation.

LLM pruning. LLM pruning methods divide along two axes: pruning granularity and saliency framework. Along the granularity axis, methods range from unstructured pruning, which scores individual weights [9, 38], to structured pruning that removes whole architectural units (channels, attention heads, or transformer blocks) for wall-clock speedups on dense hardware [25, 3, 26, 2, 22]. Along the saliency-framework axis, magnitude-based scoring [14] and gradient-based scoring [25, 27] estimate per-unit importance from weights and gradients. Layer-wise reconstruction, the framework most relevant to SKIM, scores units by how much zeroing them perturbs the layer's output on calibration activations [9, 38, 2, 22]; within this family, SparseGPT and SlimGPT additionally update the remaining weights to absorb the reconstruction error, while Wanda and FLAP do not. After pruning, most methods recover lost capability through post-hoc fine-tuning or knowledge distillation [41, 28]. SKIM extends layer-wise reconstruction by partitioning calibration activations by the role each position plays under structured generation, and pairs the position-aware saliency with a constrained-distribution distillation recipe rather than standard output-matching distillation. We accordingly select Wanda [38], FLAP [2], and SlimGPT [22] as baselines, spanning layer-wise reconstruction and providing the most direct comparison to SKIM.

LLM distillation. Knowledge distillation (KD) facilitates knowledge transfer from a teacher to a student model by matching output distributions or intermediate hidden states [16, 34]. While KD has long been a well-established paradigm in language modeling [17, 35, 39], recent research has shifted toward on-policy approaches, where students learn from their own generated trajectories to improve alignment in generative tasks [13, 1, 42]. Additionally, KD has been utilized to recover performance degradation following LLM pruning [28, 37]. However, to the best of our knowledge, no existing studies have explored KD specifically within the context of grammar-constrained decoding.

B Implementation Details

B.1 SKIM: Pruning

Per-category normalization. In practice, the per-category activation sums in the saliency formulation are averaged within each category before α-weighted aggregation. The implemented saliency score is

$$S_i^{\text{pos}} = \left(\alpha_c \cdot \frac{1}{|P_c|} \sum_{t \in P_c} X_{ti}^2 + \alpha_d \cdot \frac{1}{|P_d|} \sum_{t \in P_d} X_{ti}^2 + \alpha_f \cdot \frac{1}{|P_f|} \sum_{t \in P_f} X_{ti}^2 \right) \cdot \|W_{i,:}\|_2^2, \tag{18}$$

which differs from Eq. 11 by the $1/|P_\kappa|$ factors. Without this normalization, the most populous category (typically format) would dominate the saliency irrespective of the chosen α values: even

Table 9: Surrogate precision design comparison on Hammer-3B at 20% sparsity. Numbers are tool-use accuracy (%) after pruning and distillation under matched compute.

Surrogate precision	BFCL-v3	API-Bank
Exponential (Eq. 19)	80.3	63.7
Additive (Eq. 18)	**88.1**	**72.9**

with $\alpha_f \ll \alpha_d$, an unweighted format sum can outweigh a small but heavily weighted decision sum simply because $|P_f| \gg |P_d|$ on the calibration set. The same per-category averaging is used in the self-attention and linear-attention saliencies described below.

Surrogate precision design: additive vs. exponential. The form of $\tilde{H}_{ii}^{\text{pos}}$ is one of several ways to encode the position-level inductive bias into the diagonal Hessian. The form used throughout SKIM is the *additive* one already introduced in Eq. 18: each category contributes a non-negative weighted average of squared activations into the precision. A natural alternative is an *exponential* form that treats context and decision positions as additive precision gains and format positions as multiplicative variance injection:

$$S_i^{\text{exp}} = \left(\alpha_c \cdot \tfrac{1}{|P_c|}\textstyle\sum_{t\in P_c} X_{ti}^2 + \alpha_d \cdot \tfrac{1}{|P_d|}\textstyle\sum_{t\in P_d} X_{ti}^2\right) \cdot \exp\left(-\,2\alpha_f \cdot \tfrac{1}{|P_f|}\textstyle\sum_{t\in P_f} X_{ti}^2\right) \cdot \left\|W_{i,:}\right\|_2^2. \tag{19}$$

This decouples format positions from the additive precision pool and instead penalizes neurons whose activations concentrate on format tokens through an exponential gate.

Table 9 compares the two on Hammer-3B at 20% sparsity. The additive form is both simpler and stronger on both benchmarks, so we adopt it for all main results.

Pruning unit and saliency by module type. SKIM applies category-aware saliency at three different granularities, chosen to match the architecture of each module type in Qwen3.5. We describe the prunable unit (intermediate neuron, KV group, or linear-attention group); the saliency in Eq. 18 is applied to the input feature of the projection whose input feature index coincides with that unit.

- **MLP**. The structured unit is the intermediate MLP neuron, which appears as an output dimension of `gate_proj` and `up_proj` and as an input dimension of `down_proj`. We score neuron j by applying Eq. 18 to `down_proj`, identifying its input feature index with the neuron index. Removing neuron j zeros the corresponding output dimension of `gate_proj` and `up_proj` and the corresponding input dimension of `down_proj`.
- **Self-attention** (regular softmax attention, with `q_proj`, `k_proj`, `v_proj`, `o_proj`). The structured unit is the KV group: the head dimensions associated with one shared (K, V) pair across the query heads in that group. Eq. 18 is applied per KV group rather than per `o_proj` channel, because the group, not the channel, is the pruning unit and its query heads share one (K, V) pair: we score each KV group by the projected head-output energy $\|A_t^{(h)} W_O^{(h)}\|_2^2$, summed over the query heads in the group and averaged within each token category, then aggregated across categories with the same $(\alpha_c, \alpha_d, \alpha_f)$ as MLP pruning. Dropping a KV group zeros the corresponding output dimensions of W_Q, W_K, W_V and the corresponding input dimensions of W_O in that group.
- **Linear attention** (the `linear_attn` module in Qwen3.5's Gated DeltaNet block, architecturally distinct from regular self-attention). The structured unit is the linear-attention group: the channel block shared across the input projections, `conv1d`, and `out_proj`. We score individual channels by applying Eq. 18 to `out_proj` (identifying its input feature index with the channel index), then aggregate channel scores within each group; the group, not the channel, is the prunable unit. Dropping a group zeros the corresponding output dimensions of the input projections (and `conv1d`, if present) and the corresponding input dimensions of `out_proj` in the same layer.

Token category labels on calibration data. We classify calibration positions into context, decision, and format using a hybrid rule. *Output positions* (positions the model generates) are labeled

by their role in the call: function names, argument names and values, and the brackets and commas that set the number of calls are decision positions unless the grammar forces them ($|\mathcal{V}_t| = 1$); all other output positions are format. $|\mathcal{V}_t|$ alone does not separate the two, as the JSON grammar admits optional whitespace almost everywhere. *Input positions* are not all treated as context: only *tool-relevant* input tokens (function signatures, argument descriptions, the user query) are labeled context, and these labels are produced by prompting GPT-4 [29] over each calibration prompt and asking which spans are tool-relevant. Remaining input tokens (system boilerplate, role markers, special tokens) are excluded from P_c, P_d, P_f entirely and do not contribute to any saliency sum.

Context labels without GPT-4. Table 7 compares the GPT-4 labels with two alternatives that need no proprietary model: a tool-span rule that labels as context the spans holding tool or function names, parameter names, parameter descriptions, and the user query, which selects about 26% of the prompt tokens, and labeling every prompt token as context. The rule matches GPT-4 labeling on Qwen3.5-2B and trails it by 1.5–4.6 points on Qwen3.5-9B, whereas labeling the full prompt as context degrades accuracy substantially on Qwen3.5-9B, so identifying the relevant spans matters more than preserving every token of the prompt, and the rule is a reproducible alternative.

Hyperparameters and procedure. For all main results we use $\alpha_c = 1.0$, $\alpha_d = 1.0$, $\alpha_f = 0.2$, held fixed across model and target sparsity. The calibration set is the same xLAM-FC [23] subset used by the layer-wise reconstruction baselines. Pruning is one-shot in the Wanda sense [38]: for each module we compute the saliency once, prune the lowest-scoring units to that module's target sparsity, and do not perform sequential weight updates as in SparseGPT [9]. The per-layer sparsity allocation across the network follows FLAP [2]; SKIM's contribution is in the saliency, not in this allocation: Table 10 shows that giving every layer the same budget performs comparably.

Table 10: Pruning schedule on Qwen3.5-2B and BFCL-v3 (three further seeds, mean ± std.).

Schedule	25% sparsity	50% sparsity
Adaptive across layers (default)	79.8 ± 1.7	64.8 ± 0.9
Uniform across layers	78.7 ± 0.9	66.7 ± 0.7

Sensitivity to the category weights. Table 11 compares three weightings of $(\alpha_c, \alpha_d, \alpha_f)$ on a random subset of 100 BFCL-v3 items, so the values are comparable only within the table. The context- and decision-heavy weighting is best at every scale and sparsity, followed by the uniform weighting and then the format-heavy weighting (the two tie at 25% on Qwen3.5-2B); the key choice is to prioritize context and decision positions over format positions rather than one specific tuple.

Table 11: Structured accuracy on a 100-item subset of BFCL-v3 under three category weightings.

	Qwen3.5-2B		Qwen3.5-4B	Qwen3.5-9B
Weighting $(\alpha_c, \alpha_d, \alpha_f)$	25%	50%	50%	50%
Context- and decision-heavy $(2.0, 2.0, 0.5)$	**80.0**	**69.0**	**79.0**	**81.0**
Uniform $(1.0, 1.0, 1.0)$	79.0	54.0	75.0	77.0
Format-heavy $(0.5, 0.5, 2.0)$	79.0	47.0	73.0	74.0

B.2 SKIM: Distillation

Loss. The full SKIM training objective combines the on-policy constrained-KL term (Eq. 15) with a position-aware hidden-state matching term:

$$\mathcal{L}_{\text{SKIM}} = \mathcal{L}_{\text{KD}} + \hat{\lambda}_H \mathcal{L}_{\text{hidden}}, \tag{20}$$

where

$$\mathcal{L}_{\text{hidden}} = \sum_{\ell \in L} \sum_t \hat{\alpha}_{\text{cat}(t)} \left\| h_t^{(p,\ell)} - h_t^{(q,\ell)} \right\|_2^2, \qquad \hat{\alpha}_c = \frac{\alpha_c}{\sum_{c'} \alpha_{c'}}, \tag{21}$$

and $h_t^{(p,\ell)}, h_t^{(q,\ell)}$ are the teacher and student hidden states at position t following transformer block ℓ. The matched layer set L contains three transformer block outputs spaced every other layer, i.e., $L = \{\ell_0, \ell_0 + 2, \ell_0 + 4\}$ for a chosen ℓ_0. The per-position weights $\hat{\alpha}_{\text{cat}(t)}$ form a probability distribution over the three categories ($\sum_c \hat{\alpha}_c = 1$), so $\hat{\alpha}$ controls only the *relative* importance of

context, decision, and format positions while the scalar $\hat{\lambda}_H$ controls the overall weight of the hidden-state objective. The unnormalized weights $(\alpha_c, \alpha_d, \alpha_f) = (1.0, 1.0, 0.2)$ are reused exactly from the pruning saliency, giving $(\hat{\alpha}_c, \hat{\alpha}_d, \hat{\alpha}_f) = (5/11, 5/11, 1/11)$; we use $\hat{\lambda}_H = 1.1$ throughout, and we match raw L2 distances without per-layer normalization or cosine terms.

On-policy distillation (OPD). Off-policy distillation evaluates both models only on ground-truth prefixes $y^*_{<t}$. Under structured decoding, a decision that differs from the ground truth, such as another function name, makes the grammar follow that function's schema, so the following positions and valid sets $\mathcal{V}_t$ do not appear in the ground-truth sequence. A pruned student often makes such decisions and must continue from them at deployment. On-policy distillation [1, 13] trains the student on its own responses; SKIM generates them under the grammar, so the constrained KL of Eq. 15 covers the decision positions and valid sets the deployed student reaches.

Phase 1 – off-policy warm-up (first 25% of optimizer steps). Targets use the ground-truth completions y^* from the xLAM-FC subset: the teacher distribution $p_\phi(\cdot \mid x, y^*_{<t})$ is evaluated by teacher-forcing the ground-truth $y^*_{<t}$, and the student matches this distribution under $\mathcal{L}_{\text{KD}}$ and $\mathcal{L}_{\text{hidden}}$ at the same positions; this warm-up gives stable targets before the student sees its noisy early rollouts.

Phase 2 – on-policy (remaining 75% of optimizer steps). For each x, the current student generates y of $\mathcal{L}_{\text{KD}}$ (up to 300 new tokens) greedily from q_ϕ instead of sampling, as at evaluation, under the example's JSON-schema grammar (XGrammar), with end-of-sequence masked until the grammar can terminate; an example whose grammar fails to compile or whose rollout is empty falls back to its ground-truth completion. Categories and $\mathcal{V}_t$ are recomputed on the rollout, where the teacher is teacher-forced and both losses are computed; no gradient flows through it (the $\texttt{sg}$ in $\mathcal{L}_{\text{KD}}$).

Hyperparameters. All models share the same training hyperparameters. Training runs for 2 epochs at a learning rate of 1×10^{-5}, with effective batch size 4 (4 gradient accumulation steps) and maximum sequence length 1,300 tokens. Distillation uses a 4096-sample subset of xLAM-FC [23], matching the subset used for pruning calibration. The distillation temperature is 2.0, applied to both teacher and student logits before computing $D_{\text{KL}}(p_\phi \,\|\, q_\phi)$. We run all experiments on NVIDIA A6000 (48 GB VRAM), A100 (80 GB VRAM), and H200 (141 GB VRAM) GPUs, with 256 GB system RAM and 1 TB local NVMe storage per node. With two A100 GPUs, we can run a single SKIM experiment which takes around 6–12 hours end to end, depending on the model size.

Mixing with vanilla distillation. Table 6 adds the vanilla output-matching distillation of Eq. 12 to the SKIM objective with a coefficient β on Qwen3.5-2B at 25% sparsity. Pure SKIM has the highest structured accuracy and no free-generation accuracy, vanilla distillation alone the opposite, and intermediate coefficients give operating points in between.

C Further experiments

C.1 Motivating experiment

To complement the post-hoc error analysis in Figure 2 (right), we run a controlled ablation on MLP neurons. We restrict the experiment to MLP pruning, since MLP intermediate channels admit a clean per-neuron specialization score; the resulting evidence is therefore specifically about the MLP pathway. For each MLP intermediate neuron i in each transformer layer, we measure its mean squared activation on tokens of each category $c \in \{\text{context}, \text{decision}, \text{format}\}$ over the calibration set:

$$A_i^c = \frac{1}{|\mathcal{T}_c|} \sum_{t \in \mathcal{T}_c} h_i(t)^2, \tag{22}$$

where $h_i(t)$ is the i-th intermediate activation at token position t and $\mathcal{T}_c$ is the set of calibration positions of category c. The category-c specialization score of neuron i is then

$$s_i^c = \frac{A_i^c}{\sum_{c'} A_i^{c'}}, \tag{23}$$

and we call neuron i *category-c specialized* if $s_i^c \geq \theta$ with $\theta = 0.6$, i.e., more than 60% of its activation budget on the calibration set concentrates on a single category. For each category c, we

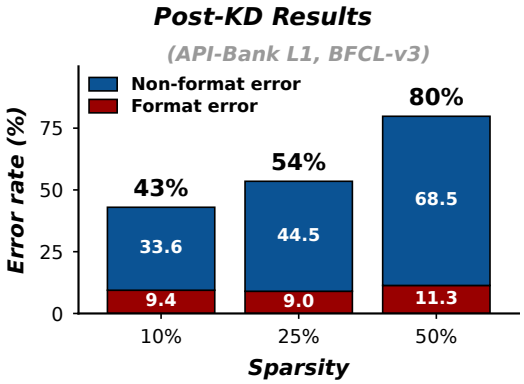


Figure 5: Pruned-then-distilled model performance averaged over three baselines, two benchmarks, and Qwen3.5-0.8B/2B (§ 3.2). Pruned baseline error grows with sparsity, but non-format error dominates: capacity is misallocated to format behavior the constraint mechanism enforces for free.

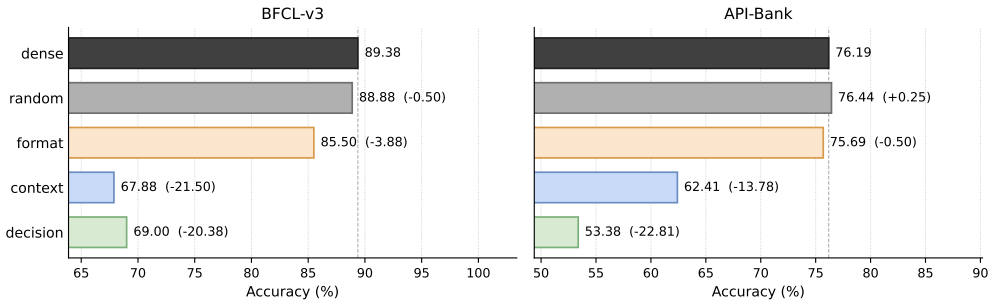


Figure 6: Task accuracy under category-specialized MLP ablation on BFCL-v3 (left) and API-Bank L1 (right) on Hammer2.1-3B model. *context*, *decision*, and *format* ablate MLP neurons whose activation budget on the calibration set concentrates ($s_i^c \geq 0.6$) on the corresponding token category; *random* ablates the same number of MLP neurons uniformly at random; *dense* is the unpruned baseline. Format-specialized ablation barely affects accuracy, while context- or decision-specialized ablation causes sharp drops in accuracy relative to the dense baseline.

then zero the corresponding down-projection rows for all category-c-specialized neurons across all layers. *random* ablates an equal number of random MLP neurons; *dense* is the unpruned model.

Figure 6 reports task accuracy on BFCL-v3 and API-Bank. Ablating format-specialized neurons keeps accuracy within $\sim$4 points of random on both benchmarks, while ablating context-specialized neurons drops accuracy by 13.8 to 21.5 points and ablating decision-specialized neurons drops accuracy by 20.4 to 22.8 points. These results provide direct evidence that format-specialized MLP capacity is largely dispensable when structured decoding handles formatting, whereas context- and decision-specialized MLP capacity is essential for task accuracy and must be preserved by pruning.

C.2 Pruning analysis for tool-expert model

If SKIM's position-aware advantage exploits role specialization at the neuron level, one expects more such specialization in a model that has been fine-tuned specifically for tool calling than in a general-purpose model. Hammer-2.1-3B is a tool-specifically fine-tuned LLM in the same parameter range as Qwen3.5-2B, and we contrast the two directly using the specialization criterion of §C.1, namely $s_i^c \geq \theta = 0.6$ for at least one token category c.

Figure 7 reports the aggregate fraction of MLP neurons specialized for each category across the entire network. The most striking gap is in the decision category: 15.7% of Hammer-2.1-3B's MLP neurons are decision-specialized, against only 6.8% in Qwen3.5-2B. Most of the difference comes out of the mixed pool (57.4% in Hammer vs. 65.7% in Qwen3.5-2B), with context- and format-specialized fractions roughly comparable across the two models. The tool-specific fine-tuning regime appears to convert mixed neurons into decision-specialized ones, consistent with the

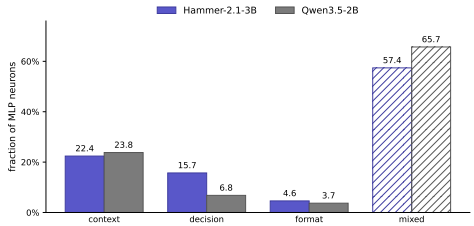


Figure 7: Aggregate fraction of MLP neurons specialized for each token category on Hammer2.1-3B and Qwen3.5-2B. *context*, *decision*, *format* correspond to context, decision, and format positions respectively; *mixed* denotes neurons that do not satisfy the $\theta = 0.6$ threshold for any single category.

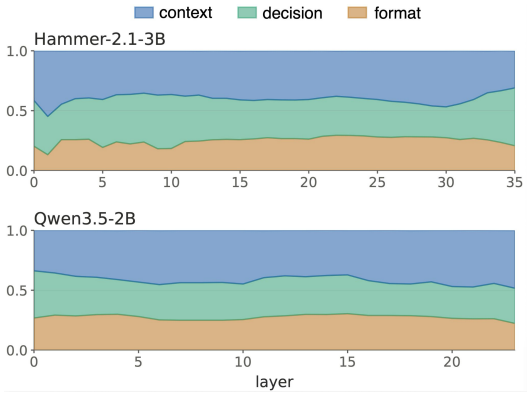


Figure 8: Layer-wise fraction of MLP neurons assigned to each token category by their largest selectivity score on Hammer-2.1-3B (top) and Qwen3.5-2B (bottom); unlike Fig. 7, every neuron is assigned to one category. At most layers, Hammer-2.1-3B has the larger decision fraction.

intuition that a function-calling objective rewards neurons that respond selectively at constrained branch points, where the format admits more than one continuation.

Figure 8 resolves this picture by depth. At most layers beyond the first few, Hammer2.1-3B sits above Qwen3.5-2B on the decision share, recapitulating the aggregate pattern at the layer level.

The implication for SKIM is that the position-aware saliency has a larger pool of clearly specialized neurons to work with on tool-expert models. This is consistent with Table 2: SKIM's margin over baselines on Hammer2.1-3B is largest at 50% sparsity, where the budget is tight enough that correctly identifying which neurons can be removed (the format-specialized ones) and which must be preserved (the context- and decision-specialized ones) determines the outcome.

C.3 Memory Efficiency

We report SKIM's memory footprint on Qwen3.5-2B, measured on physically compacted models in which the masked neurons and groups have been removed from the weight tensors. Table 12 reports parameter count, whole-model sparsity, and GPU memory at two points: *at load* (after the compacted model has been loaded onto the GPU) and *peak* (the maximum observed during a representative inference run). *Whole-model sparsity* is the realized parameter reduction across the full model: the nominal setting applies only to the prunable projections (MLP and self/linear-attention modules), so embeddings, lm-head, and untouched layers pull the realized number down. At 50% nominal sparsity, SKIM removes 34.14% of the parameters, which reduces both the load-time and the peak GPU memory by roughly 1.2 GiB at both points.

C.4 Throughput

Table 13 reports decoding throughput and prefill speed relative to the dense model at 50% nominal sparsity, measured on one 48 GB A6000 for the 2B and 9B models and on two for the 27B model.

Table 12: Memory footprint of Qwen3.5-2B under SKIM, measured on physically compacted models. *At load* is the GPU memory after weight loading; *peak* is the maximum GPU memory observed at any point during one representative inference run.

	Params	Whole-model sparsity	GPU memory at load	Peak GPU memory
Dense	1.88 B	–	7.010 GiB	7.358 GiB
SKIM-10%	1.75 B	6.77%	6.792 GiB	7.130 GiB
SKIM-25%	1.56 B	17.07%	6.420 GiB	6.754 GiB
SKIM-50%	1.24 B	34.14%	5.821 GiB	6.151 GiB

Table 13: Speedups over the dense model at 50% nominal sparsity.

Model	Parameters	Decode, batch 1	Decode, batch 256	Prefill, 4,096 tokens
Qwen3.5-2B	1.88B → 1.24B	1.03×	1.29×	1.23×
Qwen3.5-9B	8.95B → 6.06B	1.13×	1.41×	1.37×
Qwen3.5-27B	26.9B → 16.7B	1.17×	1.59×	1.51×

Width pruning keeps the sequential depth, so single-request speedups are modest, and the gains grow with model size and with batch size, as expected for width pruning.

C.5 Seed variation and larger scale

Table 14 repeats the Qwen3.5-2B comparison of Table 1 on BFCL-v3 over three further random seeds, and Table 15 repeats SKIM on Qwen3.5-4B and 9B and adds Qwen3.5-27B, for which no baselines were run. In Table 1 the free-generation accuracy of SlimGPT on Qwen3.5-9B at 50% sparsity (62.5) slightly exceeds its structured accuracy (59.1); over three seeds the two values are tied within variation (structured 60.0 $\pm$ 3.1, free 60.1 $\pm$ 4.8). Constrained decoding changes the token space available to the model rather than correcting its prediction, so it can occasionally exclude a semantically acceptable output of unconstrained decoding.

Table 14: Structured accuracy of each method on Qwen3.5-2B and BFCL-v3 over three further random seeds. Each cell reports the mean $\pm$ standard deviation across the seeds.

Method	25% sparsity	50% sparsity
Wanda-sp	64.9 $\pm$ 1.0	26.4 $\pm$ 1.3
SlimGPT	74.9 $\pm$ 0.7	47.8 $\pm$ 0.4
FLAP	76.8 $\pm$ 0.7	54.7 $\pm$ 0.6
SKIM (ours)	**79.8** $\pm$ **1.7**	**64.8** $\pm$ **0.9**

Table 15: Structured accuracy of SKIM on BFCL-v3 over three further random seeds (mean $\pm$ std.).

Model	25% sparsity	50% sparsity
Qwen3.5-2B	79.8 $\pm$ 1.7	64.8 $\pm$ 0.9
Qwen3.5-4B	84.7 $\pm$ 1.2	76.3 $\pm$ 0.6
Qwen3.5-9B	84.7 $\pm$ 0.6	77.3 $\pm$ 3.5
Qwen3.5-27B	89.0 $\pm$ 0.0	78.0 $\pm$ 1.0

C.6 Decoding temperature

All results use greedy decoding. Table 16 reports SlimGPT on Qwen3.5-9B at 50% sparsity under small positive temperatures; a temperature of 0.02 already accounts for most of the loss.

Table 16: Structured accuracy of SlimGPT on Qwen3.5-9B at 50% sparsity (BFCL-v3) under greedy decoding and three small positive decoding temperatures.

Temperature τ	0 (greedy)	0.02	0.05	0.10
Structured accuracy	59.1	56.9	55.3	55.3

C.7 Multi-step healing

Table 17 adds one epoch of healing on 128 HotpotQA trajectories to every 25%-sparsity model of Table 3; the dense row is the dense model fine-tuned on the same trajectories. Table 18 reports the same step for SKIM on Qwen3.5-2B and 9B together with its BFCL-v3 accuracy, and a variant that instead continues healing on more multi-call trajectories sampled from the xLAM-FC calibration data. Task-matched trajectory supervision brings SKIM close to the fine-tuned dense model at both scales, while its BFCL-v3 accuracy drops by at most 4.5 points from the single-turn model.

Table 17: HotpotQA F1 on Qwen3.5-2B at 25% sparsity before and after one epoch of healing on 128 HotpotQA trajectories; the dense row is the dense model fine-tuned on the same trajectories.

Method	Single-turn healing	+ HotpotQA trajectories
Dense	43.6	60.2
Wanda-sp	14.0	26.4
FLAP	14.5	30.7
SlimGPT	19.5	32.5
SKIM (ours)	**20.2**	**59.7**

Table 18: Multi-step healing of SKIM at 25% sparsity on Qwen3.5-2B and 9B. BFCL-v3 columns report structured accuracy and HotpotQA columns report F1.

	Qwen3.5-2B		Qwen3.5-9B	
Model	BFCL-v3	HotpotQA	BFCL-v3	HotpotQA
Dense, fine-tuned on HotpotQA trajectories	–	60.2	–	75.8
SKIM, single-turn healing	80.8	20.2	85.5	–
SKIM, + more multi-call trajectories	78.3	34.5	84.0	66.2
SKIM, + HotpotQA trajectories	78.7	59.7	81.0	73.5

D Prompts and Examples

We hereby attach below the full prompts (including system prompts) and the output examples for representative success and failure cases on all three datasets: BFCL-v3, API-Bank L1, and HotpotQA. For the two tool-calling datasets (BFCL-v3 and API-Bank L1), the function-call format example and the <IMPORTANT> reminder block in the system prompt are identical across cases and have been replaced with [... format example and <IMPORTANT> reminders elided ...] in each figure for compactness. The full text of the elided block appears below.

```
If you choose to call a function ONLY reply in the following format
with NO suffix:

<tool_call>
<function=example_function_name>
<parameter=example_parameter_1>
value_1
</parameter>
<parameter=example_parameter_2>
This is the value for the second parameter
that can span
multiple lines
</parameter>
</function>
</tool_call>

<IMPORTANT>
Reminder:
- Function calls MUST follow the specified format
- Required parameters MUST be specified
- You may provide reasoning in natural language BEFORE the function
  call, but NOT after
- If there is no function call available, answer normally
</IMPORTANT>
```

Figure 9: Function-call format example and <IMPORTANT> reminder block elided from the per-case figures. The block is token-for-token identical across all tool-calling cases.

Full prompt

```
<|im_start|>system
# Tools

You have access to the following functions:

<tools>
{"type": "function",
 "function": {
   "name": "protein_info.get_sequence_and_3D",
   "description": "Retrive the sequence and 3D models of proteins.",
   "parameters": {
     "type": "object",
     "properties": {
       "protein_name": {
         "type": "string",
         "description": "The name of the protein."},
       "model_3d": {
         "type": "boolean",
         "description": "Set true to get 3D model of the protein.",
         "default": true}},
     "required": ["protein_name"]}}}
</tools>

[... format example and <IMPORTANT> reminders elided ...]
<|im_end|>
<|im_start|>user
Get the protein sequence of human HbA1c, normal hemoglobin,
and rat hemoglobin and their 3D models<|im_end|>
<|im_start|>assistant
<think>
```

Expected tool call

```
[{"name": "protein_info.get_sequence_and_3D",
  "arguments":
    {"protein_name": "human HbA1c",
     "model_3d": true}},
 {"name": "protein_info.get_sequence_and_3D",
  "arguments":
    {"protein_name": "normal hemoglobin",
     "model_3d": true}},
 {"name": "protein_info.get_sequence_and_3D",
  "arguments":
    {"protein_name": "rat hemoglobin",
     "model_3d": true}}]
```

SKIM (free decoding)

```
{
  "name": "human HbA1c",
  "model_3d": true,
  "protein_name": "human HbA1c",
  "model_3d": true,
  "model_3d": true,
  "model_3d": true,
  ...
}
```

SKIM + XGrammar

```
[{"name": "protein_info.get_sequence_and_3D",
  "arguments":
    {"protein_name": "human HbA1c",
     "model_3d": true}},
 {"name": "protein_info.get_sequence_and_3D",
  "arguments":
    {"protein_name": "normal hemoglobin",
     "model_3d": true}},
 {"name": "protein_info.get_sequence_and_3D",
  "arguments":
    {"protein_name": "rat hemoglobin",
     "model_3d": true}}]
```

Baseline (Wanda-SP)

```
{"name": "protein_info_get_sequence_and_3D",
 "arguments": {
   "protein_name": "hA1c",
   "model_3d": true}}
```

Figure 10: BFCL-v3 success case. SKIM under free decoding collapses into a malformed loop that mixes tool-call fields with argument values; XGrammar restores the exact parallel call set. Wanda-SP at equal density emits one call with a corrupted function name and a truncated `protein_name`.

Full prompt

```
<|im_start|>system
# Tools

You have access to the following functions:

<tools>
{"type": "function",
 "function": {
   "name": "calculate_derivative",
   "description":
     "Calculate the derivative of a polynomial function.",
   "parameters": {
     "type": "object",
     "properties": {
       "function": {
         "type": "string",
         "description": "The polynomial function."},
       "x_value": {
         "type": "number",
         "description":
           "The x-value at which the derivative is calculated.
            Optional, default to 0.00."}},
     "required": ["function"]}}}
</tools>

[... format example and <IMPORTANT> reminders elided ...]
<|im_end|>
<|im_start|>user
Calculate the derivative of the function 3x^2 + 2x - 1.<|im_end|>
<|im_start|>assistant
<think>
```

Expected tool call

```
{"name": "calculate_derivative",
 "arguments": {
   "function": "3x**2 + 2x - 1",
   "x_value": 0.0}}
```

SKIM (free decoding)

```
{
  "function": "3x^2 + 2x - 1",
  "arguments": {
    "function": "3x^2 + 2x - 1",
    "arguments": {
      "arguments": {
        ...
      }
    }
  }
}
```

SKIM + XGrammar

```
{"name": "calculate_derivative",
 "arguments": {
   "function": "3x^2 + 2x - 1",
   "x_value": 0.0}}
```

Baseline (Wanda-SP)

```
{"name": "calculate_derivative",
 "arguments": {
   "function": "3x^2 + 2 + - 1",
   "x_value": 3.02}}
```

Figure 11: BFCL-v3 failure case. Grammar enforcement repairs the structure (well-formed JSON with the right function name and argument keys), but the model writes `3x^2` where the gold uses `3x**2`, so the case fails exact match despite format validity. Free decoding loops; Wanda-SP corrupts the polynomial expression in the generated call.

Full prompt

```
<|im_start|>system
# Tools

<tools>
{"type": "function",
 "function": {
   "name": "ModifyPassword",
   "description":
     "API for modifying the password of the account.",
   "parameters": {"type": "object",
     "properties": {
       "token":        {"type": "string"},
       "old_password": {"type": "string"},
       "new_password": {"type": "string"}},
     "required": ["token", "old_password", "new_password"]}}}
{"type": "function",
 "function": {
   "name": "GetUserToken",
   "description":
     "Get the user token by username and password.",
   "parameters": {"type": "object",
     "properties": {
       "username": {"type": "string"},
       "password": {"type": "string"}},
     "required": ["username", "password"]}}}
</tools>

[... format example and <IMPORTANT> reminders elided ...]
<|im_end|>
<|im_start|>user
Can you help me change my password?<|im_end|>
<|im_start|>assistant
Sure, I can help you with that. Can you please provide your
username and current password?<|im_end|>
<|im_start|>user
My username is user1 and my current password is user1pass.<|im_end|>
<|im_start|>assistant
Thank you. Before changing your password, we need to generate
a token to authenticate your account. Let me call GetUserToken.
Generate API Request:<|im_end|>
<|im_start|>assistant
<think>
```

Expected tool call

```
{"name": "GetUserToken",
 "arguments": {
   "username": "user1",
   "password": "user1pass"}}
```

SKIM (free decoding)

```
{
  "username": "user1",
  "password": "user1pass"
]
}
</think>
{
  "username": "user1",
  "password": "user1pass"
]
}
...
```

SKIM + XGrammar

```
{"name": "GetUserToken",
 "arguments": {
   "username": "user1",
   "password": "user1pass"}}
```

Baseline (Wanda-SP)

```
{"name": "user1",
 "arguments": {
   "username": "user1",
   "password": "user1pass"}}
```

Figure 12: API-Bank L1 success case. The multi-turn dialog requires the assistant to authenticate before modifying the password; SKIM with XGrammar emits the correct `GetUserToken` call. Free decoding loses the function-name slot; Wanda-SP places the username in the `name` field.

Full prompt

```
<|im_start|>system
# Tools

<tools>
{"type": "function",
 "function": {
   "name": "DeleteAccount",
   "description": "Delete an account.",
   "parameters": {"type": "object",
     "properties": {"token": {"type": "string"}},
     "required": ["token"]}}}
{"type": "function",
 "function": {
   "name": "GetUserToken",
   "description":
     "Get the user token by username and password.",
   "parameters": {"type": "object",
     "properties": {
       "username": {"type": "string"},
       "password": {"type": "string"}},
     "required": ["username", "password"]}}}
</tools>

[... format example and <IMPORTANT> reminders elided ...]
<|im_end|>
<|im_start|>user
Can you help me delete my account?<|im_end|>
<|im_start|>assistant
Sure, I can help you with that. Can you please provide your
username and password?<|im_end|>
<|im_start|>user
My username is user3 and my password is user3pass.<|im_end|>
<|im_start|>assistant
Thank you. For security purposes, I need to authenticate your
account first. Just a moment please.
Generate API Request:<|im_end|>
<|im_start|>assistant
<think>
```

Expected tool call

```
{"name": "GetUserToken",
 "arguments": {
   "username": "user3",
   "password": "user3pass"}}
```

SKIM (free decoding)

```
{
  "username": "user3",
  "password": "user3pass",
  "token": "your_token_here",
  "token": "your_token_here",
  "username": "user3",
  "password": "user3pass",
  ...
}
```

SKIM + XGrammar

```
{"name": "DeleteAccount",
 "arguments": {
   "token": "your_token_here"}}
```

Baseline (Wanda-SP)

```
{"name": "user3",
 "arguments": {
   "username": "user3",
   "password": "user3pass"}}
```

Figure 13: API-Bank L1 failure case. The grammar-constrained decoder produces a perfectly valid call to `DeleteAccount` with a placeholder `token`; the user is still in the authentication step and the expected call is `GetUserToken`. The format is right, the tool is wrong.

Question

Who is younger, Wayne Coyne or Toshiko Koshijima?

Gold answer

Toshiko Koshijima

SKIM trajectory

```
Thought: Open[2] and Open[8] to check their birth years.
  Next Action: Open[2]
Action: Open[2]
Observation: [2] Wayne Coyne: Wayne Michael Coyne (born
  January 13, 1961) is an American musician. He is the lead
  singer, occasional backing vocalist, guitarist, keyboardist,
  theremin player and songwriter for the band the Flaming Lips.

Thought: Open[8] to check Toshiko Koshijima's birth year.
  Next Action: Open[8]
Action: Open[8]
Observation: [8] Toshiko Koshijima: Toshiko Koshijima (born
  March 3, 1980 in Kanazawa, Ishikawa) is a Japanese singer.
  Along with composer Yasutaka Nakata, she is a lead vocalist
  of the electronica band Capsule, formed in 1997.

Thought: 3, 1980. 1961. 1980. 1961. 1980. 1961. ...
Action: Open[1]
Observation: [1] Capsule (band): Capsule is a Japanese
  electronica band consisting of record producer Yasutaka
  Nakata and vocalist Toshiko Koshijima.

Thought: 3, 1980. 1961. 1980. 1961. 1980. 1961. ...
Action: Open[3]
Observation: [3] Gummy Song Fetus: a three-track EP consisting
  of a USB drive embedded inside a gummy fetus, hand-delivered
  by Wayne Coyne to a record store in Lawrence, Kansas.
```

SKIM final answer

Toshiko Koshijima

Baseline answer (Wanda-SP)

Wayne Coyne

Metrics (EM / F1)

EM = 1.0, F1 = 1.0

Figure 14: HotpotQA success case. The retrieval trajectory is noisy (the model loops through `Open[1]` and `Open[3]` after already having both birth years), but SKIM converges to the correct span; EM = 1.0, F1 = 1.0. Wanda-SP returns the wrong person.

Question

The Order of the Lithuanian Grand Duke Gediminas features one of the earliest symbols of Lithuania, that was used by the Lithuanian Republic as a minor state symbol during what period?

Gold answer

between World War I and World War II

SKIM trajectory

```
Thought: Open[4]
Action: Open[4]
Observation: [4] Order of the Lithuanian Grand Duke Gediminas:
  the Lithuanian Presidential Award, re-instituted to honour
  citizens of Lithuania for outstanding performance in civil
  and public offices. It features the Columns of Gediminas,
  one of the national symbols of Lithuania.

Thought: The evidence states that the Order features the
  Columns of Gediminas, one of the national symbols of
  Lithuania. It does not specify the period during which the
  Lithuanian Republic used this symbol.
Action: Open[1]
Observation: [1] Columns of Gediminas: The Columns of Gediminas
  or Pillars of Gediminas are one of the earliest symbols of
  Lithuania and one of its historical coats of arms. During the
  period between World War I and World War II they were used
  by the Lithuanian Republic as a minor state symbol.

Thought: The evidence for [1] states that they were used by
  the Lithuanian Republic as a minor state symbol during the
  period between World War I and World War II. Therefore, the
  answer is: between World War I and World War II.
Action: Finish
Observation: Ready to answer.
```

SKIM final answer

World War I and World War II

Baseline answer (Wanda-SP)

Order of the Lithuanian Grand Duke Gediminas

Metrics (EM / F1)

EM = 0.0, F1 = 0.9333

Figure 15: HotpotQA failure case. The evidence chain is correct and the model's own reasoning quotes the gold span verbatim, but the final answer drops the leading preposition "between"; the answer scores EM = 0.0 and F1 = 0.9333 against the gold span.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and § 1 state our contributions; the structured-decoding gains widening at higher sparsity and the structured-vs-free trade-off claimed there are supported by the experiments in § 5.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: § 6 discusses limitations on model scale (up to 27B in the Qwen3.5 series, with full baseline comparisons up to 9B, and Hammer-2.1-3B), distillation data (single-turn function-calling traces only), and pruning-method scope (layer-wise reconstruction, not yet extended to end-to-end pruning or quantization).

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper presents an empirical compression method and does not contain formal theorems or proofs requiring an assumption set. However, refer to § 4 for theoretically-motivated proposed method.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: § 5 specifies the models, datasets, sparsity levels, baselines, and key hyperparameters; § B contains the full hyperparameter list and implementation specifics needed for reproduction.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code with configuration files for the main experiments is available on the project page (`https://moon4sake.github.io/skim`); the corresponding setup and reproduction instructions are described in § 5 and § B.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: § 5 reports training and evaluation details (data splits, sparsity levels, calibration corpus, distillation hyperparameters, evaluation benchmarks, and decoding settings); full details are deferred to § B.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: Tables 10, 14 and 15 report mean and standard deviation over three seeds for the pruning schedule, the main comparison on Qwen3.5-2B, and SKIM up to 27B. Compute cost prevented running multiple seeds for every (model scale, sparsity level, method) cell; the remaining cells are single runs, and robustness there is supported by trend consistency across four model scales, three sparsity levels, and two backbones in § 5.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: § B reports the GPU type and the per-run training cost (e.g., epochs, kd budget, etc).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The work conforms with the NeurIPS Code of Ethics: it uses publicly released models and datasets under their original licenses, does not involve human subjects, and does not raise specific safety, fairness, or privacy concerns beyond those inherent to general LLM compression research.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A].

    Justification: Our work is a foundational compression method for existing tool-calling LLMs, and its societal impact is mediated via the underlying models, with no direct application path that introduces additional positive or negative impacts beyond those of the base models being compressed.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A].

    Justification: The paper does not release pretrained models or scraped datasets that pose a high risk of misuse.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All used models (Qwen3.5 series, Hammer-2.1-3B), datasets (xLAM-FC, BFCL-v3, API-Bank L1, HotpotQA, Spider, GSM8K), and baseline implementations (Wanda, FLAP, SlimGPT) are cited in § 5. All their respective licenses are respected.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The code released on the project page is documented with running bash scripts including training/evaluation matching the experiments in § 5.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A].

    Justification: The work does not involve crowdsourcing or human-subject research.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A].

    Justification: The work does not involve human-subject research; no IRB approval was required.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: GPT-4 is used to label tool-related context tokens in the calibration corpus, as described in § 5 and § B. This is a labeling shortcut, not a methodological requirement: a rule-based selection of tool-related spans matches GPT-4 labeling on Qwen3.5-2B and trails it by 1.5–4.6 points on Qwen3.5-9B (Table 7).