PROFF: PROMPT-ADAPTIVE REDUCED-ORDER FEED-FORWARD COMPUTATION FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern LLM agents repeatedly process long prompts containing system instructions, tool schemas, and interaction history while generating short outputs. This makes prefill a major inference bottleneck, with feed-forward (FF) blocks accounting for a large fraction of its compute. Existing FF acceleration methods largely rely on neuron-level sparsity, which is less effective for modern gated activations such as SwiGLU and GeGLU. We present **PROFF** (**P**rompt-adaptive **R**educed-**O**rder **F**eed-**F**orward computation), which replaces the standard FF block with a reduced-order FF block. PROFF compresses the hidden state into a low-dimensional space, performs mixing in that space, and projects it back to the original hidden dimension. A lightweight predictor selects the reduced dimension $P^\star$ for each agent call, adapting FF compute to the input. The surrogate is trained offline through per-layer distillation, while the remaining model parameters are frozen. Across four LLMs and multiple agentic benchmarks, PROFF retains near-baseline task quality while improving prefill throughput by 2.15–2.23$\times$ across prompt lengths and model architectures. Across multi-call agent workloads, PROFF further achieves 1.98–2.31$\times$ lower end-to-end task latency. These results show that prompt-conditioned reduced-order FF computation provides a practical approach to accelerating prefill-heavy agentic inference.

1 INTRODUCTION

Large language models (LLM) are increasingly deployed in agentic pipelines, where autonomous agents invoke tools, call sub-agents, and chain reasoning steps to complete multi-step tasks (Patil et al., 2025). This deployment mode has rapidly become the dominant operating regime: production agents have been observed sending $150{,}000$+ input tokens for as few as 29 tokens of structured output, leaked Claude system prompts measure $\sim$24,000 tokens (Silfverskiöld, 2026), and benchmarks of real-world agentic instructions report an average of $\sim$1,700 tokens per call (Qi et al., 2025). Across all of these, the per-call ratio of input tokens to output tokens routinely exceeds 10^2–10^3, the inverse of the chatbot regime that historically shaped LLM serving infrastructure.

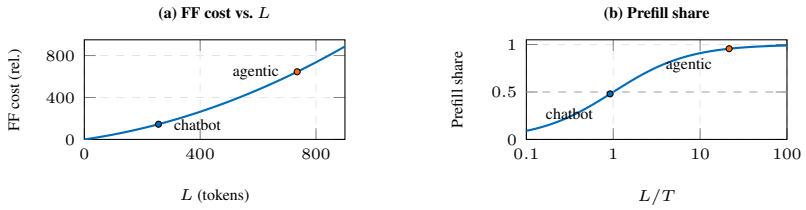

Figure 1: **Agentic workloads are prefill-dominated.** Here, L is the input length and T is the output length. **(a)** FF prefill cost increases with input length L (Zhong et al., 2024; Agrawal et al., 2024). **(b)** Chatbot calls spend roughly half of FF compute on prefill, while agentic calls spend $\sim$96% due to long inputs and short outputs (Luo et al., 2025).

Classical chatbot workloads typically balance prefill and decode, splitting feed-forward (FF) compute across both phases. Agentic workloads shift this balance: they repeatedly process long input contexts while generating only short, structured outputs. As shown in Figure 1, FF prefill cost grows

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

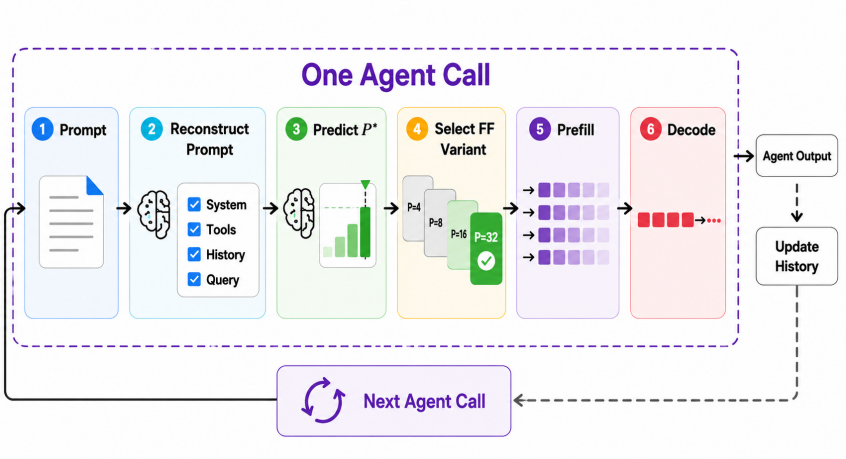

Figure 2: PROFF workflow for multi-call agent inference. Each call predicts $P^\star$, selects the corresponding FF variant, and executes prefill and decode before updating the agent history.

linearly with input length, and its share of compute increases sharply as inputs become longer than outputs. While chatbot calls spend roughly half of their FF compute on prefill, agentic calls can spend $\sim$96%. Thus, FF prefill compute, rather than decoding, becomes the primary per-request bottleneck in modern agentic deployments.

Much work targets agentic inference cost, but most of it never touches the FF block itself. It operates at one of three layers. *Application-level* methods shorten the input: prompt caching, semantic caching, dynamic tool search, model routing, and context compaction all send fewer tokens to the model (Silfverskiöld, 2026). *System-level* methods make each operator faster without changing what it computes: KV-cache management, paged attention, FlashAttention, and weight-only quantization (Frantar et al., 2023; Dettmers et al., 2023). Both layers treat the FF block as a black box. The *algorithmic* layer changes the FF block's mathematical form, and until recently it relied on *linear sparsity*: finding neurons whose activations are zero or near-zero, then skipping the associated weight loads (Liu et al., 2023; Alizadeh et al., 2024). Gated activations broke this lever. When SwiGLU and GeGLU replaced ReLU, the fraction of zero-valued activations fell from $\sim$95% in OPT-175B to under 10% in LLaMA-3-8B and Gemma-2 (Mirzadeh et al., 2024). Restoring ReLU costs tens to hundreds of billions of tokens of continued pretraining and does not help architectures whose gating cannot be removed. Other linear methods inherit the same problem: L_1 pruning and static low-rank decomposition (Lehtimäki et al., 2024b) see only per-neuron patterns, and modern gated FF blocks have little per-neuron structure to find.

Unlike attention-side methods such as StreamingLLM (Xiao et al., 2024a) and MInference (Jiang et al., 2024), which accelerate an unchanged pretrained model by exploiting existing attention structure, **PROFF** (**P**rompt-adaptive **R**educed-**O**rder **F**eed-**F**orward computation) modifies the model's FF computation itself. We replace each standard FF block with a distilled reduced-order substitute and dynamically select its capacity $P^\star$ for each agent call. Rather than selecting which components of the pretrained FF block to execute, PROFF learns a lower-dimensional FF representation that is not present in the original checkpoint.

PROFF replaces the standard FF block with a reduced-order block: compress $H \to P$, mix in the reduced space $P \to P$, and project back $P \to H$, where $P \ll H$ and $P \ll H'$ (Figure 3). Building on this design, PROFF (Figure 2) selects an appropriate reduced dimension $P^\star$ for each agent call, reducing FF computation while preserving the standard model interface. Our contributions are:

- **Structured-prompt characterization**: an empirical characterization showing that structured agent prompts (System / Tools / History / Query) exhibit more compressible FF representations, motivating prompt-conditioned compute adaptation.
- **Reduced-order FF computation**: a family of pre-compiled reduced-order FF blocks that replace the standard FF computation with a lower-dimensional $H \to P \to P \to H$ computation. PROFF

improves prefill throughput by 2.15–2.23× and reduces end-to-end agent latency by 1.98–2.31× while retaining near-baseline task quality.

- **Lightweight per-call dimension predictor**: a small MLP that maps the structured prompt representation to an appropriate reduced dimension $P^\star$ before model execution, enabling prompt-adaptive FF compute with negligible prediction overhead.

2 METHOD

2.1 PROMPT RECONSTRUCTION

Agent frameworks typically construct prompts from structured components such as System, Tools, History, and Query, although these components may be serialized into free-form text before inference. Prompt reconstruction recovers this scaffold, exposing prompt structure to the P-predictor before model execution. Figure 2 shows reconstruction as the preprocessing stage of each agent call. To illustrate the difference, consider the same function-calling prompt in unstructured and structured forms:

Unstructured Agent Prompt:
I'm helping a user book a flight from Boston to Tokyo for under $1500, arriving Thursday morning. I already found three options and ranked them. Now the user says they actually need the redeye instead of the morning flight, so I have to fix the departure date and account for the time-zone shift. What tool should I call next?

Structured Agent Prompt:

System You are a flight-booking assistant with access to search and booking tools. Always confirm time-zone and date before booking. **### Tools** search_flights(origin, dest, date, max_price); rank_by(criteria, k); confirm_booking(option_id). **### History** User requested flight Boston to Tokyo, budget $1500, arriving Thursday morning. Three options were returned and ranked by price and duration. **### Query** "Wait, I just realized I need the redeye, not the morning flight. Adjust departure date and timezone."

Both prompts convey the same information, but the structured form exposes surface features (number of tools, length of history, query type) observable before any forward pass. The unstructured form folds the same content into continuous narrative, leaving the FF block's effective rank inferrable only through in-network inspection. To bring unstructured prompts into this regime, we apply an online LLM pass that reconstructs each free-form prompt into the scaffold.

Table 1: Per-window activation statistics for Unstructured and Structured prompts. **Cosine sim.**: cosine similarity (± std) of consecutive-layer activations. **Linear / Nonlinear Eig.**: number of components needed to explain 95% of activation variance under SVD / KPCA.

	Cosine sim.		Linear Eig.		Nonlinear Eig.	
Window	**Unstr.**	**Struct.**	**Unstr.**	**Struct.**	**Unstr.**	**Struct.**
T25-30/L25-30	.361±.024	.736±.022	129	92	145	102
T5-10/L5-10	.357±.025	.728±.026	158	111	168	121
T15-20/L15-20	.343±.025	.737±.024	155	124	169	137
T25-30/L10-15	.357±.026	.735±.025	132	103	150	121

Table 1 reports activation statistics on agent prompts from BFCL v3, computed at windows that pair a band of token positions with a band of LFM2-2.6B decoder layers (`T`a`--`b`/L`c`--`d = tokens a–b at layers c–d, $0 \le c, d \le 31$). We measure three *aligned* windows at early (`T5--10/L5--10`), middle (`T15--20/L15--20`), and late (`T25--30/L25--30`) depths, plus one *cross-aligned* window (`T25--30/L10--15`) pairing late tokens with mid-depth layers to rule out token-position or depth-specific artifacts.

The statistics in Table 1 are computed over n=500 prompts per category. **Cosine similarity** measures adjacent-layer activation similarity, where higher values indicate greater inter-layer redundancy. **Linear Eig.** and **Nonlinear Eig.** report the number of components required to explain

95% of activation variance using SVD and Kernel PCA, respectively; lower values indicate greater compressibility.

Structured prompts show substantially higher inter-layer similarity ($\sim$0.73 vs. $\sim$0.35) and lower effective rank under both linear (92–124 vs. 129–158) and nonlinear (102–137 vs. 145–169) representations. This indicates that structured prompts concentrate activations in a smaller subspace, making them more amenable to reduced-order projection. This structure provides observable cues that enable the predictor to estimate an appropriate $P^\star$ from prompt features without runtime activation inspection.

2.2 P-PREDICTOR AND RUNTIME DISPATCH

Prompt-conditioned compute budget. Different prompts induce FF representations with different degrees of compressibility. As shown in §2.1, structured prompts exhibit lower effective activation rank, suggesting that they can often be represented accurately with a smaller reduced dimension. We therefore treat P as a prompt-conditioned compute budget rather than using a fixed dimension for every call.

Let $\mathcal{P} = \{4, 8, 16, 32\}$ denote the set of available pre-compiled dimensions. For a prompt x, we define the oracle dimension as the smallest P whose reduced-order FF output satisfies a fixed reconstruction-quality constraint:

$$P_{\text{oracle}}(x) = \min_{P \in \mathcal{P}} \{P : \mathcal{E}_P(x) \leq \epsilon_{\text{rec}}\},$$

where $\mathcal{E}_P(x)$ measures the reconstruction error between the original FF output and its rank-P surrogate, and ϵ_{rec} is a fixed quality threshold. We select ϵ_{rec} on held-out calibration data by sweeping candidate thresholds and choosing the largest tolerance that satisfies the predefined reconstruction-quality target. The selected threshold is then fixed for all downstream evaluations. Thus, more compressible prompts can be assigned smaller dimensions, whereas prompts requiring greater representational capacity are assigned larger dimensions.

Learning and runtime selection. Computing $P_{\text{oracle}}(x)$ online would require evaluating multiple candidate FF variants and would eliminate the benefit of adaptive execution. Instead, PROFF learns this selection rule offline. The reconstructed System/Tools/History/Query prompt is encoded into a representation z_x, which exposes observable features such as tool structure, history length, and query form. A lightweight MLP is trained using $P_{\text{oracle}}(x)$ as the supervision target and produces a categorical distribution

$$p_\theta(P \mid z_x), \qquad P \in \mathcal{P}.$$

At inference time, PROFF selects

$$\widehat{P}(x) = \arg\max_{P \in \mathcal{P}} p_\theta(P \mid z_x).$$

The selected $\widehat{P}(x)$ determines the corresponding pre-compiled reduced-order FF variant, which is used throughout prefill and decode for that call. In this way, prompt structure serves as an inexpensive proxy for FF compressibility, allowing PROFF to adapt the FF compute budget without inspecting runtime activations or evaluating multiple candidate dimensions.

2.3 REDUCED-ORDER FF BLOCK

Once the predictor (§2.2) has chosen a reduced dimension P for a call, the agent executes the call at that dimension via a family of pre-compiled reduced-order FF variants (Figure 3): copies of the base model that differ in the capacity of their feed-forward layer.

We write H for the model's hidden dimension, H' for the intermediate dimension of the standard FF block, and P for the reduced dimension selected per call, with $P \ll H$ and $P \ll H'$. A standard FF block maps $H \to H' \to H$ with $O(HH')$ per-token computational cost. PROFF replaces it with a reduced-order $H \to P \to P \to H$ block:

- **Compress ($H \to P$):** a learned gated encoder E_P maps the hidden state into a P-dimensional latent representation.

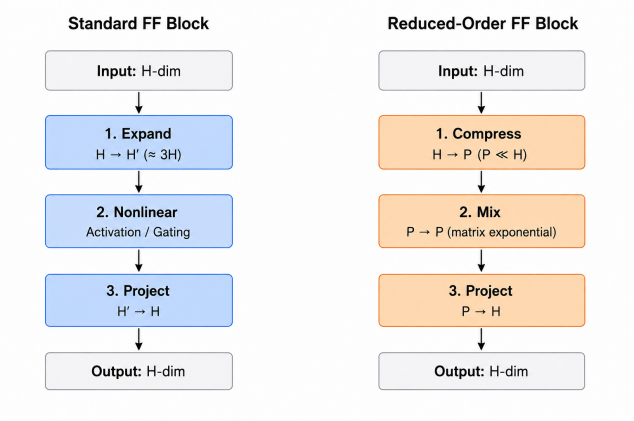

Figure 3: Reduced-order FF block. The standard FF layer (left) is replaced by a three-stage block (right): compress $H \to P$, mix $P \to P$, project $P \to H$.

- **Mix ($P \to P$):** the reduced dynamics are represented by $M_P = e^{A_P \tau}$. Since A_P is fixed after distillation, M_P is precomputed once at model-load time. Runtime mixing therefore requires a $P \times P$ matrix multiplication with $O(P^2)$ per-token cost.
- **Project ($P \to H$):** a learned decoder D_P projects the reduced representation back to the original hidden dimension, preserving compatibility with the remaining model layers.

Compute reduction. A standard FF block has per-token cost $O(HH')$, whereas PROFF requires $O(HP+P^2)$ for compression, reduced-space mixing, and projection. For $P \ll H, H'$, the resulting FF-level reduction therefore scales approximately as H'/P. This factor describes the reduction in FF computation, not whole-model speedup; attention, normalization, residual operations, and other runtime costs remain unchanged.

2.4 Reduced-Order Approximation Rationale

We model the nonlinear FF map through its induced operator $\mathcal{K}$ on a space of observables. Although the FF map F_τ is nonlinear in the hidden state, the induced operator is linear in the lifted observable space. A rank-P projection of this space yields reduced latent dynamics that can be implemented by PROFF's compress–mix–project block.

The following analysis provides an approximation rationale for the reduced-order FF block rather than asserting that every FF mapping admits an accurate rank-P representation. Motivated by the low-dimensional activation structure observed in §2.1, we consider the regime in which the prompt-induced FF dynamics admit a dominant reduced subspace. The spectral-gap assumption formalizes the existence of a dominant low-dimensional component separated from discarded modes; it is consistent with, but not implied by, the reduced effective ranks observed empirically in Table 1.

Theorem 1 (Reduced-order approximation)**.** *Let $F_\tau : \mathcal{X}_0 \to \mathcal{X}$ denote the FF map over a finite horizon $\tau \le \tau_{\max}$, and let Π_P be a rank-P projection onto a dominant reduced subspace of the induced operator $\mathcal{K}$. Assume that the projected dynamics admit a reduced linear generator A_P, and let $E_P : \mathcal{X}_0 \to \mathbb{R}^P$ and $D_P : \mathbb{R}^P \to \mathcal{X}$ denote the encoder and decoder implemented by the reduced-order FF block. Define*

$$\widehat{F}_P(h) = D_P\big(e^{A_P \tau} E_P(h)\big)\,.$$

Suppose E_P and D_P are Lipschitz on the operating region with constants L_E and L_D. Then, under the finite-horizon and spectral gap assumptions, there exist constants $C_1, C_2 > 0$ such that

$$\left\| F_\tau(h) - \widehat{F}_P(h) \right\| \le L_D C_1 L \tau e^{-\delta\tau} + L_D C_2 \sigma_{P+1}(\mathcal{K}) + \varepsilon_{\mathrm{enc}}(P) + \varepsilon_{\mathrm{dec}}(P), \qquad h \in \mathcal{X}_0,$$

where L is the Lipschitz constant of the underlying dynamics, $\delta > 0$ is the spectral gap, and $\sigma_{P+1}(\mathcal{K})$ is the first singular value of $\mathcal{K}$ excluded by the rank-P projection. The singular values are ordered as $\sigma_1 \ge \sigma_2 \ge \cdots$.

Here, Π_P denotes the ideal rank-P projection used in the analysis, whereas the learned encoder E_P need not implement Π_P exactly. The term $\varepsilon_{\mathrm{enc}}(P)$ accounts for this mismatch, while $\varepsilon_{\mathrm{dec}}(P)$ captures the corresponding reconstruction error introduced by the learned decoder. Thus, increasing P reduces the spectral truncation term, while the learned encoder and decoder determine how closely the implemented block realizes the ideal reduced dynamics.

The theorem matches the implemented PROFF block: the compressor E_P is nonlinear, the reduced dynamics are linear, and the decoder D_P maps the latent state back to the original hidden dimension.The result bounds the reduced-order approximation error and explicitly separates spectral truncation from the approximation errors of the learned encoder and decoder. It does not directly bound downstream task loss.

The prompt-level error $\mathcal{E}_P(x)$ used for oracle dimension selection in §2.2 measures the realized discrepancy between the original and reduced FF outputs. The bound above identifies the sources that contribute to this discrepancy: spectral truncation, reduced-dynamics residuals, and learned encoder–decoder approximation errors.

In PROFF, E_P, A_P, and D_P are learned offline by minimizing the per-layer reconstruction loss on a calibration corpus. The theorem therefore provides the approximation rationale for the reduced latent dynamics, while empirical distillation determines how closely the learned block realizes the ideal representation. A proof sketch is provided in Appendix 6.1.

3 Experiments and Results

3.1 Experimental Settings

Hardware. Experiments are performed on three NVIDIA RTX A6000 GPUs (48 GB GDDR6 each; 144 GB aggregate GPU memory). This platform is used for both offline knowledge distillation and inference evaluation across all models and PROFF configurations.

Datasets. We distill offline on FineWeb-Edu (Penedo et al., 2024) at sequence length 2048; it is not used for evaluation. For capability evaluation we use GPQA Diamond (Rein et al., 2024) as an out-of-distribution probe and AID (Banerjee & Gupta, 2025), which supplies matched structured and unstructured renderings of the same prompts. For agentic evaluation we use BFCL v3 (Patil et al., 2025), τ^2-bench (Barres et al., 2025), and AgentBoard (Ma et al., 2024), on which we additionally measure end-to-end task latency and calls per task. Prefill throughput is measured across prompt lengths. Splits, item counts, and scoring protocols are given in Appendix 6.2.

Base models. We evaluate four open-weight models: LFM2-2.6B (Liquid AI, 2025) (hybrid convolution + attention), Llama-3-8B (Dubey et al., 2024) (dense), GPT-OSS-20B (OpenAI, 2025) (MoE), and Qwen3-30B-A3B (Yang et al., 2025) (MoE). For each model, we replace the FF layer with the reduced-order block; attention, embedding, normalization, and the LM head are retained from the original checkpoint and frozen.

Baselines. We compare against three classes of baseline. (i) *Unreduced base models*, run under the same serving stack and hardware as PROFF. (ii) *Serving-level optimization*: vLLM with PagedAttention (Kwon et al., 2023) and TensorRT-LLM (NVIDIA, 2024b), which accelerate execution without altering the model's mathematical form. (iii) *Algorithmic FF methods*: DejaVu (Liu et al., 2023) for contextual sparsity, and the static compression methods SVD-LLM (Wang et al., 2025) and SliceGPT (Ashkboos et al., 2024), both configured at compression ratios matched to PROFF's $P{=}16$ setting so that quality is compared at equal parameter reduction.

3.2 Main Results

3.2.1 Ablation Study.

PROFF reduces test time for both unstructured and structured prompts. In contrast, LFM2+DejaVu provides little improvement, with speedups of only $1.02\times$ and $1.01\times$, respectively.

On structured AID, replacing the linear ODE parameterization (PROFF-LODE) increases test loss from 3.86 to 3.92 while reducing the speedup from $2.33\times$ to $2.26\times$. Replacing the reduced-order

mixing block with 1024 GRU cells (PROFF-1024) similarly increases test loss to 3.97 and achieves a lower $2.28\times$ speedup, indicating that additional recurrent capacity does not recover the quality–efficiency balance of the reduced-order design. Applying sparse NN acceleration to the dense layer (PROFF+sparse) increases the speedup to $2.36\times$, but also increases test loss to 4.02. PROFF-LODE+sparse achieves the highest speedup at $2.38\times$, but incurs the largest test loss (4.07) among the reduced-order variants.

Key takeaway. Structured prompts expose more compressible representations, allowing PROFF to achieve greater inference-time savings. At the same time, near-baseline task accuracy is preserved, demonstrating that the efficiency gains introduce minimal quality degradation.

Table 2: Inference ablation study of PROFF on Unstructured and Structured AID. **TeLoss**: test reconstruction loss, reported as mean ± bootstrap SE. **TeT**: test time (s), reported as mean ± standard deviation across repeated runs. **Sp**: inference speedup over the corresponding LFM2 baseline. Lower is better for TeLoss and TeT; higher is better for Sp. Best in **bold**, second-best underlined.

	Unstructured AID			Structured AID		
Model	**TeLoss**	**TeT**	**Sp**	**TeLoss**	**TeT**	**Sp**
LFM2	4.20 ± 0.12	569 ± 16	$1.00\times$	4.05 ± 0.11	431 ± 13	$1.00\times$
LFM2+DejaVu	4.28 ± 0.12	557 ± 15	$1.02\times$	4.12 ± 0.11	426 ± 12	$1.01\times$
PROFF	$\mathbf{4.00 \pm 0.11}$	273 ± 8	$2.08\times$	$\mathbf{3.86 \pm 0.10}$	185 ± 6	$2.33\times$
PROFF-LODE	$\underline{4.07 \pm 0.11}$	282 ± 8	$2.02\times$	$\underline{3.92 \pm 0.10}$	191 ± 6	$2.26\times$
PROFF-1024	4.10 ± 0.11	278 ± 8	$2.05\times$	3.97 ± 0.10	189 ± 6	$2.28\times$
PROFF+sparse	4.15 ± 0.12	$\underline{270 \pm 8}$	$\underline{2.11\times}$	4.02 ± 0.10	$\underline{183 \pm 6}$	$\underline{2.36\times}$
PROFF-LODE+sparse	4.18 ± 0.12	$\mathbf{267 \pm 8}$	$\mathbf{2.13\times}$	4.07 ± 0.11	$\mathbf{181 \pm 6}$	$\mathbf{2.38\times}$

3.2.2 Comparison with Compression Baselines

Table 3 compares PROFF with static model-compression baselines on BFCL v3. PROFF achieves a $2.10\times$ inference speedup, outperforming SVD-LLM ($1.28\times$) (Wang et al., 2025) and SliceGPT ($1.38\times$) (Ashkboos et al., 2024). PROFF also retains 68.7% BFCL accuracy, compared with 70.0% for the unreduced baseline. Overall, PROFF provides the best quality–efficiency tradeoff among the evaluated compression methods.

Table 3: Comparison with model-compression baselines on BFCL v3 using GPT-OSS-20B under matched hardware and serving settings. Decode latency is reported as mean ± standard deviation across repeated runs. **Sp** denotes speedup over the unreduced GPT-OSS-20B baseline.

Method	**BFCL v3 (%) ↑**	**Decode Lat. (ms/tok) ↓**	**Sp ↑**
GPT-OSS-20B (unreduced)	$\mathbf{70.0 \pm 1.2}$	12.34 ± 0.41	$1.00\times$
SVD-LLM	67.8 ± 1.3	9.62 ± 0.34	$1.28\times$
SliceGPT	66.9 ± 1.3	8.91 ± 0.31	$1.38\times$
PROFF	$\underline{68.7 \pm 1.2}$	$\mathbf{5.88 \pm 0.27}$	$\mathbf{2.10\times}$

3.2.3 Benchmarks

Across all four base models, PROFF exhibits a consistent quality–capacity tradeoff as the reduced dimension P increases. Smaller dimensions, particularly $P{=}4$, incur the largest performance degradation, while $P{=}16$ substantially recovers quality and $P{=}32$ generally provides the closest match to the unreduced baseline. This trend is consistent across both single-turn and agentic benchmarks, showing diminishing quality gains as the reduced dimension increases.

Adaptive P provides a complementary operating point by selecting the reduced dimension on a per-call basis rather than applying the same capacity to every prompt. Its performance generally falls between the $P{=}16$ and $P{=}32$ configurations and remains close to the unreduced baseline, with small variations across models and tasks. Overall, the results show that PROFF can substantially reduce FF capacity while preserving near-baseline capability, while prompt-adaptive dimension selection avoids uniformly using the largest reduced dimension for every call.

Table 4: Benchmark across single-turn and agentic benchmarks. Fixed-P configurations characterize the quality–capacity tradeoff, while **Adaptive** P uses the per-call dimension selected by the prompt-conditioned predictor. Values are mean ± bootstrap SE over evaluation items. Within each base model, best in **bold**, second-best underlined.

Base	Config	GPQA-D (%)	AID (%)	BFCL v3 (%)	τ^2-bench pass^4	AB-PR (%)	AB-GA (%)
LFM2-2.6B	Base	**35.0** ± 1.2	**71.2** ± 1.8	61.5 ± 1.4	54.8 ± 1.9	58.4 ± 2.0	63.1 ± 1.9
	P=4	31.2 ± 1.3	66.5 ± 1.9	55.0 ± 1.5	48.2 ± 2.0	53.1 ± 2.1	58.0 ± 2.0
	P=8	33.0 ± 1.2	69.0 ± 1.8	58.5 ± 1.4	51.0 ± 1.9	55.7 ± 2.0	60.5 ± 1.9
	P=16	34.4 ± 1.2	70.5 ± 1.8	60.8 ± 1.4	54.0 ± 1.9	57.8 ± 2.0	62.4 ± 1.9
	P=32	35.1 ± 1.2	71.1 ± 1.8	**61.7** ± 1.4	**55.0** ± 1.9	**58.6** ± 2.0	**63.3** ± 1.9
	Adaptive P	34.7 ± 1.3	70.8 ± 1.7	61.2 ± 1.4	54.5 ± 1.8	58.1 ± 1.9	62.8 ± 1.8
Llama-3-8B	Base	**34.2** ± 1.3	**73.0** ± 1.7	**66.0** ± 1.3	**57.2** ± 1.8	**61.5** ± 1.9	**67.0** ± 1.8
	P=4	29.8 ± 1.4	67.1 ± 1.8	59.8 ± 1.4	49.8 ± 1.9	55.0 ± 2.0	61.0 ± 1.9
	P=8	32.1 ± 1.3	70.0 ± 1.7	63.0 ± 1.4	53.2 ± 1.9	58.7 ± 1.9	64.2 ± 1.8
	P=16	33.5 ± 1.3	72.0 ± 1.7	65.0 ± 1.3	56.1 ± 1.8	60.5 ± 1.9	66.1 ± 1.8
	P=32	34.0 ± 1.3	72.8 ± 1.7	65.8 ± 1.3	57.0 ± 1.8	61.2 ± 1.9	66.8 ± 1.8
	Adaptive P	33.8 ± 1.3	72.4 ± 1.7	65.3 ± 1.4	56.6 ± 1.8	60.9 ± 1.9	66.4 ± 1.8
Qwen3-30B-A3B	Base	62.0 ± 1.2	**79.0** ± 1.5	60.2 ± 1.3	**60.0** ± 1.7	**68.0** ± 1.7	**72.0** ± 1.7
	P=4	56.5 ± 1.3	72.8 ± 1.6	54.0 ± 1.4	52.5 ± 1.8	61.0 ± 1.8	65.5 ± 1.8
	P=8	59.3 ± 1.2	75.8 ± 1.5	57.0 ± 1.3	56.2 ± 1.7	64.5 ± 1.7	68.8 ± 1.7
	P=16	61.2 ± 1.2	77.8 ± 1.5	59.2 ± 1.3	58.8 ± 1.7	67.0 ± 1.7	71.0 ± 1.7
	P=32	**62.1** ± 1.2	78.7 ± 1.5	**60.5** ± 1.3	59.7 ± 1.7	67.8 ± 1.7	71.7 ± 1.7
	Adaptive P	61.8 ± 1.3	78.2 ± 1.5	60.1 ± 1.3	59.2 ± 1.8	67.4 ± 1.8	71.3 ± 1.7
GPT-OSS-20B	Base	**49.0** ± 1.3	**82.0** ± 1.4	**70.0** ± 1.2	**63.0** ± 1.7	**71.0** ± 1.7	**75.0** ± 1.6
	P=4	40.0 ± 1.5	75.0 ± 1.5	62.0 ± 1.3	54.0 ± 1.8	63.0 ± 1.8	68.0 ± 1.7
	P=8	43.0 ± 1.4	78.0 ± 1.5	65.0 ± 1.3	57.0 ± 1.7	66.0 ± 1.8	70.5 ± 1.7
	P=16	44.5 ± 1.4	80.5 ± 1.4	68.7 ± 1.2	60.5 ± 1.7	69.2 ± 1.7	73.3 ± 1.6
	P=32	47.0 ± 1.3	81.7 ± 1.4	69.2 ± 1.2	62.0 ± 1.7	70.5 ± 1.7	74.4 ± 1.6
	Adaptive P	46.0 ± 1.4	80.9 ± 1.4	68.9 ± 1.3	61.5 ± 1.8	70.0 ± 1.8	73.8 ± 1.7

3.2.4 Latency Across Agent Workloads

Table 5 reports prefill, decode, and end-to-end task latency across four model architectures and three agentic workloads. PROFF consistently reduces latency in both model execution phases, with prefill speedups ranging from $2.08\times$ to $2.45\times$ and decode speedups from $2.03\times$ to $2.39\times$.

These gains translate into $1.98\times$–$2.31\times$ lower end-to-end task latency across BFCL multi-turn, AgentBoard, and τ^2-bench. End-to-end task latency measures the wall-clock time from the initial request to the final task response, including all LLM calls, prompt reconstruction, and tool execution within the agent trajectory. For example, a task may require one LLM call to select a tool and additional calls to process the tool result and produce the final response. The end-to-end gains are therefore smaller and more variable than the isolated prefill and decode improvements because PROFF does not accelerate all components of the agent workflow.

3.2.5 Architecture and Prompt-Length Robustness

Table 6 evaluates PROFF across different model architectures and prompt lengths. PROFF consistently improves prefill throughput by 2.15–$2.23\times$ over the baseline across all configurations. The gains remain stable as prompt length increases from 10 to 1K tokens and extend across both dense and MoE-based architectures, indicating that PROFF's acceleration is not specific to a particular model architecture or prompt length.

3.2.6 Effect of reduced dimension P.

We sweep PROFF's reduced dimension P over $\{4, 6, 8, 12, 16, 24, 32\}$ on LFM2-2.6B evaluated on BFCL v3 and report test loss and distillation time in Figure 4. Increasing P improves approximation quality at the cost of additional computation. Test loss decreases from 0.090 at P=4 to 0.057 at P=16, a 36.7% reduction, and reaches 0.045 at P=32. Relative to the total observed loss reduction between P=4 and P=32, P=16 captures approximately 73% of the improvement, indicating diminishing returns as the reduced dimension increases.

Distillation time remains approximately 672 s for $P \leq 8$ and then increases with P, reaching 705 s at P=32. Thus, larger reduced dimensions improve test loss while incurring additional computa-

Table 5: Prefill, decode, and end-to-end agent latency across models and workloads. All models are evaluated on BFCL multi-turn, AgentBoard, and τ^2-bench. Prefill latency is measured per request, decode latency per generated token, and end-to-end latency from the initial request to the final task response. **Sp** denotes speedup over the corresponding unreduced baseline. Best in **bold**, second-best underlined.

Model	Workload	Prefill		Decode		End-to-End	
		Base/PROFF (ms)	Sp	Base/PROFF (ms/tok)	Sp	Base/PROFF (s/task)	Sp
LFM2-2.6B	BFCL multi-turn	42.6/18.4	2.32×	5.21/2.38	2.19×	5.82/2.58	2.26×
	AgentBoard	46.8/21.5	2.18×	5.44/2.29	**2.38×**	10.84/5.47	1.98×
	τ^2-bench	44.5/18.2	**2.45×**	5.32/2.49	2.14×	8.15/3.53	**2.31×**
Llama-3-8B	BFCL multi-turn	76.8/35.4	2.17×	8.46/3.63	**2.33×**	7.36/3.35	2.20×
	AgentBoard	83.2/34.8	**2.39×**	8.81/4.34	2.03×	13.72/6.20	**2.21×**
	τ^2-bench	80.1/36.9	2.17×	8.63/3.72	2.32×	10.21/4.78	2.14×
Qwen3-30B-A3B	BFCL multi-turn	146.2/64.1	2.28×	14.72/6.25	2.36×	8.42/4.01	2.10×
	AgentBoard	158.4/69.8	2.27×	15.18/6.91	2.20×	18.65/9.42	1.98×
	τ^2-bench	151.7/65.7	**2.31×**	14.95/6.26	**2.39×**	12.38/5.51	**2.25×**
GPT-OSS-20B	BFCL multi-turn	118.6/50.7	2.34×	12.34/5.88	2.10×	7.91/3.50	**2.26×**
	AgentBoard	127.5/61.3	2.08×	12.71/5.40	**2.35×**	16.24/7.58	2.14×
	τ^2-bench	122.4/50.8	**2.41×**	12.52/5.89	2.13×	11.46/5.08	**2.26×**

Table 6: Prefill throughput (tokens/s) across model architectures and prompt lengths, reported as mean ± standard deviation across repeated runs. Values in parentheses denote speedup over the corresponding baseline.

Model	Method	Prompt Length (tokens)			
		10	64	256	1K
LFM2-2.6B	Baseline	85.3 ± 2.7	89.0 ± 2.5	91.0 ± 2.4	92.1 ± 2.3
	PROFF	**183.4 ± 5.1** (2.15×)	**193.1 ± 5.3** (2.17×)	**200.2 ± 5.2** (2.20×)	**205.4 ± 5.4** (2.23×)
Qwen3-30B-A3B	Baseline	42.3 ± 1.4	44.3 ± 1.3	45.4 ± 1.2	45.7 ± 1.3
	PROFF	**91.8 ± 2.6** (2.17×)	**97.0 ± 2.7** (2.19×)	**99.9 ± 2.6** (2.20×)	**101.5 ± 2.8** (2.22×)
GPT-OSS-20B	Baseline	14.4 ± 0.5	15.1 ± 0.5	15.2 ± 0.4	15.3 ± 0.5
	PROFF	**31.1 ± 0.9** (2.16×)	**32.9 ± 0.9** (2.18×)	**33.4 ± 0.9** (2.20×)	**34.0 ± 1.0** (2.22×)

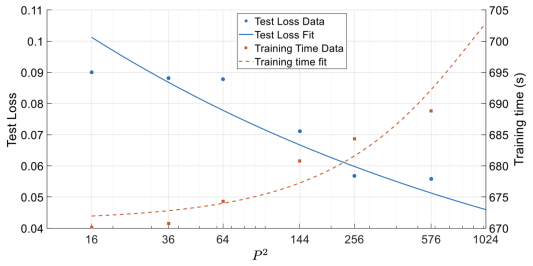

Figure 4: Reduced-dimension sweep on LFM2-2.6B evaluated on BFCL v3.

tional cost. This quality–efficiency tradeoff motivates PROFF's use of adaptive per-call dimension selection: smaller P can be used when a low-dimensional representation is sufficient, while larger P provides additional capacity when needed.

4 Conclusion

We present PROFF, a prompt-adaptive reduced-order approach for accelerating FF computation in agentic LLM inference. PROFF exploits the greater compressibility of structured prompts to select a per-call reduced dimension and execute a reduced-order FF block. Across diverse models and agentic workloads, PROFF preserves near-baseline task quality while substantially reducing prefill, decode, and end-to-end latency. These results demonstrate that prompt-conditioned reduced-order computation is a practical approach to efficient LLM agent inference.

AI USE STATEMENT

We used generative AI tools to assist with writing, editing, and improving the clarity of the paper. All technical ideas, methods, experiments, results, and claims were developed and verified by the authors. We reviewed all AI-assisted content and take responsibility for the final content of this work.

ETHICS STATEMENT

This work focuses on improving the efficiency of LLM agent inference and does not involve human subjects or the collection of private user data. We use existing public datasets and models for evaluation. Like other LLM efficiency methods, PROFF could be applied to a wide range of downstream applications, and responsible deployment should consider the risks and requirements of the specific application.

REPRODUCIBILITY STATEMENT

We provide the details needed to reproduce PROFF in the main paper and appendix. The main paper describes the reduced-order FF architecture, prompt-conditioned P-predictor, offline distillation procedure, model configurations, hardware setup, datasets, baselines, and evaluation protocols. We report results across multiple model architectures and agentic benchmarks, including fixed and adaptive reduced dimensions, latency measurements, and statistical comparisons. The appendix provides additional details on the theoretical assumptions, dataset splits and evaluation settings, predictor overhead, adaptive dimension selection, and oracle construction. We report the key hyperparameters and experimental settings used in our evaluations. Code and additional implementation details will be released to support reproduction of the reported results.

REFERENCES

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134, Santa Clara, CA, 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/agrawal.

Keivan Alizadeh, Seyed Iman Mirzadeh, Dmitry Belenko, S. Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. LLM in a flash: Efficient large language model inference with limited memory. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

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 *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2401.15024.

Ayan Banerjee and Sandeep Gupta. Personalized open world plan generation for safety-critical human centered autonomous systems: A case study on artificial pancreas. In *The 2025 Conference on Empirical Methods in Natural Language Processing*, 2025.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025. URL https://arxiv.org/abs/2506.07982.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

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.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pp. 52481–52515, 2024. URL https://arxiv.org/abs/2407.02490.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, pp. 611–626. Association for Computing Machinery, 2023. doi: 10.1145/3600006.3613165. URL https://arxiv.org/abs/2309.06180.

Mikko Lehtimäki, Lassi Paunonen, and Marja-Leena Linne. Accelerating neural ODEs using model order reduction. *IEEE Transactions on Neural Networks and Learning Systems*, 35(1):519–531, 2024a. doi: 10.1109/TNNLS.2022.3175757.

Mikko Lehtimäki, Lassi Paunonen, and Marja-Leena Linne. Accelerating neural ODEs using model order reduction. *IEEE Transactions on Neural Networks and Learning Systems*, 35(1):519–531, 2024b. doi: 10.1109/TNNLS.2022.3175757.

Liquid AI. Lfm2: Liquid foundation models, 2025. URL https://www.liquid.ai/blog/liquid-foundation-models-v2-our-second-series-of-generative-ai-models. Technical report.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Ré, and Beidi Chen. Deja vu: Contextual sparsity for efficient LLMs at inference time. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pp. 22137–22176. PMLR, 2023.

Michael Luo, Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yichuan Wang, Chi Wang, Yanping Huang, Zhifeng Chen, Joseph E. Gonzalez, and Ion Stoica. Autellix: An efficient serving engine for LLM agents as general programs. 2025. URL https://arxiv.org/abs/2502.13965.

Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1):4950, 2018. doi: 10.1038/s41467-018-07210-0.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn LLM agents. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024.

Seyed Iman Mirzadeh, Keivan Alizadeh, Sachin Mehta, Carlo C Del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. ReLU strikes back: Exploiting activation sparsity in large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

NVIDIA. TensorRT-LLM. https://github.com/NVIDIA/TensorRT-LLM, 2024a. Software library.

NVIDIA. TensorRT-LLM. https://github.com/NVIDIA/TensorRT-LLM, 2024b. Software library.

OpenAI. Introducing gpt-oss. https://openai.com/index/introducing-gpt-oss/, August 2025. Open-weight model release.

Samuel E. Otto and Clarence W. Rowley. Linearly recurrent autoencoder networks for learning dynamics. *SIAM Journal on Applied Dynamical Systems*, 18(1):558–593, 2019. doi: 10.1137/18M1177846.

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 *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392, 2025.

Guilherme Penedo, Hynek Kydlíček, Leandro von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2406.17557.

Yunjia Qi, Hao Peng, Xiaozhi Wang, Amy Xin, Youfeng Liu, Bin Xu, Lei Hou, and Juanzi Li. AgentIF: Benchmarking instruction following of large language models in agentic scenarios, 2025. URL https://arxiv.org/abs/2505.16944. NeurIPS 2025 Datasets and Benchmarks Track (Spotlight).

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level Google-proof Q&A benchmark. In *Conference on Language Modeling (COLM)*, 2024. URL https://arxiv.org/abs/2311.12022.

Ida Silfverskiöld. Agentic AI: How to save on tokens. Towards Data Science, April 2026. URL https://towardsdatascience.com/agentic-ai-how-to-save-on-tokens/. Online; accessed April 2026.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. SVD-LLM: Truncation-aware singular value decomposition for large language model compression. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2403.07378.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024a. URL https://arxiv.org/abs/2309.17453.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations (ICLR)*, 2024b.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210, Santa Clara, CA, 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin.

5 APPENDIX

5.1 LIMITATIONS

This work focuses on computational efficiency and does not provide a comprehensive assessment of the broader societal impacts. Improved efficiency may reduce computational and energy costs, but its overall impact depends on how the resulting systems are deployed and used. Issues such as environmental impact, accessibility, fairness, privacy, and responsible deployment are beyond the scope of this work and warrant further study.

6 RELATED WORK

LLM inference and serving optimization. A large body of work improves LLM inference without changing the underlying feed-forward (FF) computation. PagedAttention, implemented in vLLM, improves KV-cache memory management and serving efficiency (Kwon et al., 2023). TensorRT-LLM provides optimized kernels and runtime support for efficient LLM inference (NVIDIA, 2024a). Other work focuses on attention and long-context inference. StreamingLLM exploits attention sinks for efficient streaming generation (Xiao et al., 2024b), while MInference uses dynamic sparse attention to accelerate long-context prefilling (Jiang et al., 2024). DistServe separates prefill and decode to improve serving efficiency (Zhong et al., 2024). These methods optimize the serving system or attention computation while leaving the FF computation largely unchanged. In our experiments, we therefore compare PROFF with the unreduced base models under the same hardware and serving settings, as well as vLLM with PagedAttention and TensorRT-LLM as serving-level baselines.

Feed-forward sparsity. Another direction reduces FF computation by exploiting activation sparsity. DejaVu predicts contextual sparsity and skips inactive components during inference (Liu et al., 2023). Such neuron-level sparsity is particularly effective when many activations are zero or near zero. However, modern gated FF architectures using SwiGLU or GeGLU exhibit substantially lower activation sparsity than ReLU-based models (Mirzadeh et al., 2024), reducing the opportunity for neuron skipping. PROFF takes a different approach: instead of selecting which neurons of the original FF block to execute, it learns a lower-dimensional representation of the entire FF computation. We include DejaVu as the contextual-sparsity baseline in our evaluation.

Model compression and low-dimensional representations. Model compression provides another approach to reducing LLM inference cost. Quantization methods such as GPTQ (Frantar et al., 2023) and QLoRA (Dettmers et al., 2023) reduce the numerical precision and memory cost of model parameters. Structural and low-rank approaches instead reduce the dimensions or parameters of the model. SVD-LLM applies truncation-aware singular value decomposition for LLM compression (Wang et al., 2025), while SliceGPT removes rows and columns from weight matrices to reduce the model width (Ashkboos et al., 2024). We use SVD-LLM and SliceGPT as static compression baselines and match their compression ratios to PROFF for comparison. Unlike these static approaches, PROFF selects its reduced dimension P for each agent call, allowing FF capacity to adapt to the input.

Reduced-order computation. Model-order reduction represents high-dimensional systems using compact latent states to reduce computational cost while preserving dominant dynamics (Lehtimäki et al., 2024a). Data-driven approaches further learn low-dimensional representations directly from observations, including autoencoder-based reduced-order models and latent-space dynamics. In particular, Koopman-based methods seek coordinates in which nonlinear dynamics can be represented approximately by linear evolution (Lusch et al., 2018; Otto & Rowley, 2019). Lusch et al. learn nonlinear coordinates with approximately linear latent dynamics, while Otto and Rowley combine an autoencoder with linear recurrent dynamics to identify compact Koopman-invariant representations.

PROFF brings this reduced-order perspective to transformer FF computation. A standard $H \to H' \to H$ FF block is replaced by a learned $H \to P \to P \to H$ block, where $P \ll H, H'$. The hidden state is compressed into a low-dimensional latent representation, transformed by reduced-space dynamics, and projected back to the original hidden dimension. Unlike prior reduced-order models designed

primarily for dynamical systems, PROFF applies reduced-order computation to LLM FF blocks and treats the reduced dimension P as a prompt-conditioned compute budget. A lightweight predictor selects among pre-compiled reduced-order variants for each agent call. This combines learned reduced-order FF computation with input-dependent capacity selection, distinguishing PROFF from serving optimization, neuron-level sparsity, and static model compression.

6.1 Proof Sketch

Assumptions and interpretation. The analysis assumes that, over the finite operating horizon, the lifted FF dynamics contain a dominant rank-P subspace separated from discarded modes by a spectral gap. The projection Π_P describes this ideal reduced subspace and is used only for analysis; PROFF does not explicitly compute Π_P at inference time. Instead, the learned encoder E_P, reduced generator A_P, and decoder D_P are obtained through offline distillation. Consequently, the theorem characterizes the approximation error of an ideal reduced-order representation, while $\varepsilon_{\mathrm{enc}}(P)$ and $\varepsilon_{\mathrm{dec}}(P)$ account for discrepancies between that representation and the learned PROFF block.

Proof sketch. Let Π_P denote the rank-P projection onto the dominant subspace and let $Q_P = I - \Pi_P$. Projecting the lifted dynamics onto $\mathrm{range}(\Pi_P)$ separates the evolution into a reduced Markov term and residual contributions from the discarded subspace. Under the spectral-gap assumption, the finite-horizon memory residual satisfies

$$\|r_{\mathrm{mem}}(\tau)\| \le C_1 L \tau e^{-\delta\tau}.$$

The rank-P spectral truncation contributes

$$\|r_{\mathrm{spec}}\| \le C_2 \sigma_{P+1}(\mathcal{K}).$$

Let $z_P(h)$ denote the ideal projected latent state and let $\widetilde{z}_P(h) = E_P(h)$ be the latent state produced by the learned encoder. Similarly, let D_P denote the learned decoder. By the Lipschitz property of D_P,

$$\|F_\tau(h) - \widehat{F}_P(h)\| \le L_D\big(\|r_{\mathrm{mem}}(\tau)\| + \|r_{\mathrm{spec}}\|\big) + \varepsilon_{\mathrm{enc}}(P) + \varepsilon_{\mathrm{dec}}(P).$$

Substituting the two residual bounds gives

$$\Big\|F_\tau(h) - \widehat{F}_P(h)\Big\| \le L_D C_1 L \tau e^{-\delta\tau} + L_D C_2 \sigma_{P+1}(\mathcal{K}) + \varepsilon_{\mathrm{enc}}(P) + \varepsilon_{\mathrm{dec}}(P),$$

which is the stated result. □

6.2 Dataset Details

FineWeb-Edu. FineWeb-Edu (Penedo et al., 2024) is an educational-quality subset of the FineWeb web-crawl corpus, filtered by an educational-value classifier. We use it solely as a source of activation statistics for offline per-FF-block distillation, streaming documents at sequence length 2048 and collecting hidden states at the input and output of every decoder FF layer. No FineWeb-Edu data is used at evaluation time, so the distillation corpus is disjoint from every benchmark reported in §3.

GPQA Diamond. GPQA (Rein et al., 2024) is a graduate-level question-answering benchmark written by domain experts in biology, physics, and chemistry, constructed so that skilled non-experts with web access score well below experts. We use the Diamond split (198 questions), the highest-quality subset in which expert annotators agree and non-experts fail. GPQA is single-turn and shares no structure with our agentic workloads, so it serves as an out-of-distribution capability probe: it measures whether reduced-order FF blocks preserve reasoning ability that the distillation corpus never targeted. We report accuracy under zero-shot chain-of-thought prompting with greedy decoding (temperature 0), averaged over four independent runs.

AID. AID (Banerjee & Gupta, 2025) provides matched structured and unstructured renderings of the same underlying prompts: the two regimes differ in surface form while holding content fixed, isolating the effect of prompt structure on FF activation rank. We evaluate on 500 matched prompt pairs and report task accuracy under greedy decoding, using the identical item set for both regimes so that structured and unstructured scores are directly paired.

BFCL v3. The Berkeley Function Calling Leaderboard (Patil et al., 2025) evaluates tool-calling accuracy against real API signatures. Version 3 adds multi-turn and multi-step evaluation with state-based verification, checking both the resulting environment state and the execution path rather than string-matching the call. Its multi-turn subset contains 800 examples split evenly across four categories: Base, Missing Functions, Missing Parameters, and Long Context. We evaluate on the full 800-example multi-turn subset and score with the official state-based evaluator. BFCL v3 is also the source of the agent prompts used for the activation statistics in Table 1.

τ^2-bench. τ^2-bench (Barres et al., 2025) evaluates conversational agents in a dual-control setting where both the agent and a simulated user can act on shared environment state. It spans retail, airline, and telecom domains with tightly constrained business logic. We report pass^4, the fraction of tasks solved on all four independent attempts, which measures reliability rather than best-case success and is therefore sensitive to the quality degradation a compressed FF block might introduce.

AgentBoard. AgentBoard (Ma et al., 2024) provides multi-turn agent tasks with fine-grained progress tracking rather than binary task success. We report two metrics: progress rate (AB-PR), the fraction of subgoals reached, and grounding accuracy (AB-GA), the fraction of actions that are valid in the current environment state. Progress rate is the more informative metric for our setting because partial degradation from FF compression appears as reduced progress before it appears as outright task failure.

6.3 Statistical Data

Statistical testing. We use an approximate two-sided normal test to compare each reduced-order configuration with its corresponding unreduced baseline. Given reported means μ_{red} and μ_{Base} and bootstrap standard errors $\mathrm{SE}_{\mathrm{red}}$ and $\mathrm{SE}_{\mathrm{Base}}$, we compute

$$z = \frac{\mu_{\mathrm{red}} - \mu_{\mathrm{Base}}}{\sqrt{\mathrm{SE}_{\mathrm{red}}^2 + \mathrm{SE}_{\mathrm{Base}}^2}}, \qquad p = 2\left[1 - \Phi(|z|)\right],$$

where Φ denotes the standard normal cumulative distribution function. We report these values as approximate p-values because the test is derived from the reported bootstrap standard errors rather than from a paired resampling test.

Table 7 compares each fixed and adaptive reduced-order configuration with its corresponding unreduced base model. At $P = 4$, several quality differences are statistically significant, reflecting the capacity loss of aggressive reduction. As P increases, these differences diminish; at $P \geq 16$, most comparisons do not show a statistically significant difference from the corresponding baseline under the approximate test. The exception at $P = 16$ is GPT-OSS-20B on GPQA-D, where the difference remains significant, while no evaluated metric differs significantly at $P = 32$. Adaptive P similarly shows no statistically significant difference from the unreduced baseline on any evaluated metric, indicating that adaptive dimension selection preserves near-baseline task quality.

6.4 P-Predictor Analysis

The predictor introduces negligible overhead relative to model prefill. Its inference requires only 0.016 ms per request, compared with tens of milliseconds for prefill across the evaluated models. Thus, $P^\star$ can be selected before model execution with negligible additional latency, allowing the selected FF variant to be used throughout both prefill and decode.

6.4.1 Adaptive P Selection Across Agentic Workloads

Table 9 shows how the P-predictor adapts the reduced dimension across agent calls. Most calls are assigned $P < 32$, while a smaller fraction requires the largest dimension, showing that a fixed $P = 32$ would allocate unnecessary FF capacity to many prompts. The variation across benchmarks further indicates that PROFF adapts its compute budget to the input rather than using a fixed reduced dimension.

Table 7: Approximate two-sided p-values for each fixed and adaptive reduced-order configuration relative to its corresponding unreduced base model, computed using the normal approximation described above. Values below 0.05 are shown in **bold**. These tests are unadjusted for multiple comparisons.

Base	Config	GPQA-D	AID	BFCL v3	τ^2-bench	AB-PR	AB-GA
LFM2-2.6B	P=4	**0.032**	0.073	**0.002**	**0.017**	0.068	0.064
	P=8	0.239	0.387	0.130	0.157	0.340	0.333
	P=16	0.724	0.783	0.724	0.766	0.832	0.794
	P=32	0.953	0.969	0.920	0.941	0.944	0.941
	Adaptive P	0.865	0.872	0.880	0.909	0.913	0.909
Llama-3-8B	P=4	**0.021**	**0.017**	**0.001**	**0.005**	**0.018**	**0.022**
	P=8	0.253	0.212	0.116	0.126	0.297	0.271
	P=16	0.703	0.677	0.586	0.666	0.710	0.724
	P=32	0.913	0.934	0.913	0.937	0.911	0.937
	Adaptive P	0.828	0.803	0.714	0.814	0.823	0.814
Qwen3-30B-A3B	P=4	**0.002**	**0.005**	**0.001**	0.055	**0.005**	**0.009**
	P=8	0.112	0.131	0.082	0.362	0.145	0.183
	P=16	0.637	0.572	0.586	0.724	0.677	0.677
	P=32	0.953	0.888	0.870	0.930	0.934	0.901
	Adaptive P	0.910	0.706	0.957	0.747	0.809	0.771
GPT-OSS-20B	P=4	< 0.001	< 0.001	< 0.001	< 0.001	**0.001**	**0.003**
	P=8	**0.002**	0.051	**0.005**	**0.013**	**0.043**	0.054
	P=16	**0.019**	0.449	0.444	0.298	0.454	0.452
	P=32	0.277	0.880	0.637	0.677	0.835	0.791
	Adaptive P	0.116	0.578	0.534	0.545	0.686	0.607

Table 8: Predictor overhead relative to model prefill for a 512-token prompt. Latency is reported as mean $\pm$ standard deviation across repeated runs under the same hardware configuration.

Component	Footprint	Latency (ms)
MLP predictor	0.16 MB	0.016 ± 0.001
LFM2-2.6B prefill	4.5 GB	18.7 ± 0.6
Llama-3-8B prefill	16.1 GB	41.5 ± 1.2
GPT-OSS-20B prefill	40.2 GB	55.8 ± 1.6
Qwen3-30B-A3B prefill	60.4 GB	68.3 ± 1.9

Table 9: Distribution of dimensions selected by the P-predictor across agentic benchmarks. Entries under each P denote the percentage of agent calls assigned to that dimension. Avg. P denotes the mean selected dimension across calls.

Dataset	P=4	P=8	P=16	P=32	Avg. P
BFCL	12%	31%	42%	15%	14.48
τ^2-bench	8%	24%	46%	22%	16.64
AgentBoard	6%	20%	45%	29%	18.32

6.4.2 Effect of Prompt Structure on Adaptive P Selection

On the matched AID prompts, the predictor assigns substantially smaller dimensions to structured prompts, reducing the average selected dimension from 18.6 to 14.5 while retaining access to $P = 32$ for prompts requiring additional capacity. This provides direct evidence that prompt structure is informative for per-call FF compute selection.

Table 10: Distribution of predicted reduced dimensions on matched structured and unstructured AID prompts. Each row contains the percentage of prompts assigned to each reduced dimension by the P-predictor; **Avg.** P is the mean predicted dimension.

Prompt form	P=4	P=8	P=16	P=32	Avg. P
Unstructured	5%	20%	50%	25%	18.6
Structured	12%	31%	42%	15%	14.5
Reduction	—	—	—	—	**22.0%**

6.4.3 P-PREDICTOR ACCURACY AGAINST ORACLE SELECTION

Oracle label construction. For each calibration prompt x, we evaluate the pre-compiled PROFF variants at $P \in \{4, 8, 16, 32\}$ and compute the reconstruction error $\mathcal{E}_P(x)$ relative to the corresponding unreduced FF output. We assign $P_{\text{oracle}}(x)$ as the smallest dimension satisfying $\mathcal{E}_P(x) \leq \epsilon_{\text{rec}}$. The threshold ϵ_{rec} is selected once on held-out calibration data as described in §2.2 and is fixed before downstream evaluation. These oracle dimensions are used only as offline supervision for the P-predictor and are not computed during inference.

Table 11 shows that the P-predictor agrees with the offline oracle on 72–78% of agent calls across the evaluated workloads, while under-selecting the required dimension in only 5–8% of calls. The low under-selection rate is important because selecting a dimension below the oracle can violate the reconstruction-quality constraint, whereas over-selection primarily incurs additional computation.

Table 11: Accuracy of the P-predictor relative to oracle dimension selection. Exact match reports the fraction of calls for which $\widehat{P} = P_{\text{oracle}}$, while under-selection reports the fraction for which $\widehat{P} < P_{\text{oracle}}$.

Dataset	Exact match (%)	Under-selection (%)
BFCL	78	5
τ^2-bench	75	6
AgentBoard	72	8