Distributed Interpretability and Control for Large Language Models

Anonymous ACL submission

Abstract

Large language models that require multiple GPU cards to host are usually the most capable models. It is necessary to understand and steer these models, but the current technologies do not support the interpretability and steering of these models in the multi-GPU setting as well as the single-GPU setting. We present a practical implementation of activation-level interpretability (logit lens) and steering (steering vector) that scales up to multi-GPU language models. Our system implements design choices that reduce the activation memory by up to 7× and increase the throughput by up to 41× compared to a baseline on identical hardware. We demonstrate the method across LLaMA-3.1 (8B, 70B) and Qwen-3 (4B, 14B, 32B), sustaining 20–100 tokens/s while collecting full layer-wise activation trajectories for sequences of 1,500 tokens. Using label-position steering vectors injected post-LayerNorm, we show controllable, monotonic shifts in model outputs with a mean steerability slope of 0.702 across evaluated datasets, without fine-tuning or additional forward passes. We release detailed benchmarks, ablations, and a reproducible instrumentation recipe to enable practical interpretability and real-time behavioral control for frontier LLMs.

1 Introduction

Large language models (LLMs) now drive applications in scientific analysis, healthcare, law, and interactive decision-making, increasing the need to understand *why* a model produces a specific output and how we can reliably influence its behavior. As models scale into the tens of billions of parameters, we increasingly require interpretability tools that expose intermediate computations and enable targeted interventions during inference. Yet achieving visibility at this scale remains difficult. Interpretability methods must surface internal structure with fine granularity while preserving the throughput and memory constraints required for real-world deployment. Today, the models for which we most need interpretability are unfortunately the models for which interpretability remains least feasible.

Existing activation-level interpretability techniques including the logit lens and its variants and activation steering approaches provide conceptual insight but fail to scale to contemporary 10B+ parameter models. Prior implementations often (1) require multiple forward passes, (2) no KV caching, (3) project every intermediate activation into the full vocabulary, or (4) assume single-GPU execution. These design choices inflate memory usage, increase latency, and frequently trigger out-of-memory (OOM) failures, even on mid-sized models. As a result, activation-level interpretability and steering for large language models is effectively not possible.

In this work, we introduce a *distributed, single-pass framework* for activation-level interpretability and behavioral steering that runs at native inference speed on models up to 70B parameters. We integrate instrumentation *inside* the tensor-parallel inference path so we can capture activations during standard autoregressive decoding without extra forward passes, without interfering with KV caching, and without materializing full vocabulary logits. We instead record compact hidden-state slices for each token and perform a single batched LM-head projection only after decoding finishes. This design keeps memory proportional to the hidden dimension rather than vocabulary size, enabling full-layer, long-sequence analyses under fixed per-GPU budgets. Empirically, we reduce activation memory by up to 7× and increase throughput by up to 41× compared to a reimplemented LogitLens4LLMs (Wang, 2025) baseline on identical hardware.

We also show that this same architecture supports real-time *behavioral steering*. We compute steering vectors from base and target forward activations and inject them post-LayerNorm inside wrapped transformer layers. Because we inject in-

side the existing tensor-parallel forward path, steering preserves single-pass execution and introduces no extra compute. Across LLaMA-3.1 (8B, 70B) and Qwen3 (4B, 14B, 32B), we observe monotonic, controllable output shifts with a mean steerability slope of 0.702, demonstrating that activation-level interventions remain reliable at frontier scale without fine-tuning or throughput degradation.

Prior interpretability methods such as logit lens and activation steering are developed under single-device or offline assumptions and are fundamentally incompatible with tensor-parallel, KV-cached inference used by modern large language models. In such settings, hidden states are sharded across devices and discarded incrementally during generation, making layer-wise decoding or intervention infeasible without additional forward passes or parameter modification. We address this gap by introducing a unified execution model that integrates interpretability and steering directly into tensor-parallel inference via single-pass activation capture and deferred projection.

Contributions We summarize our contributions below:

- **Tensor-parallel interpretability.** We introduce a tensor-parallel inference architecture that enables large language models (LLaMA, Qwen) to be efficiently analyzed and controlled across multiple GPUs. By integrating logit-lens analysis and activation steering directly into tensor-parallel execution, our approach supports full-layer interpretability during standard inference, enabling scalable analysis up to 70B models at native resolution without centralizing model weights or activations.
- **Memory-efficient, single-pass instrumentation.** We combine (i) single-pass activation capture, (ii) deferred vocabulary projection, and (iii) non-redundant hidden-state logging. This design reduces activation footprint by up to 7× and sustains 20–100 tokens/s while tracing 1,500-token sequences across all layers of LLaMA and Qwen models.
- **Scalable steering vectors with real-time control.** We implement post-LayerNorm steering inside the distributed forward path, achieving stable, monotonic behavioral modifications with no additional passes and no loss in throughput. Steering remains effective across all tested model scales.

Together, these contributions close a practical gap: we make full-layer, token-level interpretability and activation-level control feasible upto 70B parameter models using commodity multi-GPU hardware, without slowing generation or exceeding memory budgets. Our results point toward unified interpretability-and-control systems suitable for both research and production.

Roadmap Section 2 reviews interpretability and steering methods. Section 3 presents our distributed, single-pass architecture. Section 4 reports scalability and steering results across five models. Section 6 discusses limitations and future directions.

2 Related Work

2.1 Logit Lens and Tuned Lens Methods

The *logit lens*, introduced by Nostalgebraist (Nostalgebraist, 2020), projects intermediate residual-stream activations through the output embedding to expose how token predictions evolve across layers. While it reveals a progression from diffuse to sharpened distributions, it suffers from *basis drift*, leading to misaligned intermediate predictions. Belrose et al. (Belrose et al., 2023) addressed this limitation with the *tuned lens*, learning a lightweight per-layer linear correction that substantially improves alignment between intermediate logits and final model outputs.

Subsequent tooling such as LogitLens4LLMs (Wang, 2025) extended these ideas to modern architectures (e.g., LLaMA and Falcon), adding improved visualization pipelines and projection-fidelity metrics. However, existing logit-lens implementations typically rely on multi-pass decoding or single-GPU execution, restricting practical applicability to models below roughly 10B parameters.

2.2 Steering Vectors and Activation Interventions

Steering vectors provide a complementary mechanism for controlling model behavior by directly modifying hidden activations rather than reading them. Turner et al. (2023) showed that a simple direction computed from positive versus negative examples can reliably steer attributes such as sentiment or topic without retraining. Compared to gradient-based approaches such as PPLM, steering vectors are lightweight, reusable, and state-based.

Recent work has focused on robustness and scalability. (Tan et al., 2024) systematically evaluated

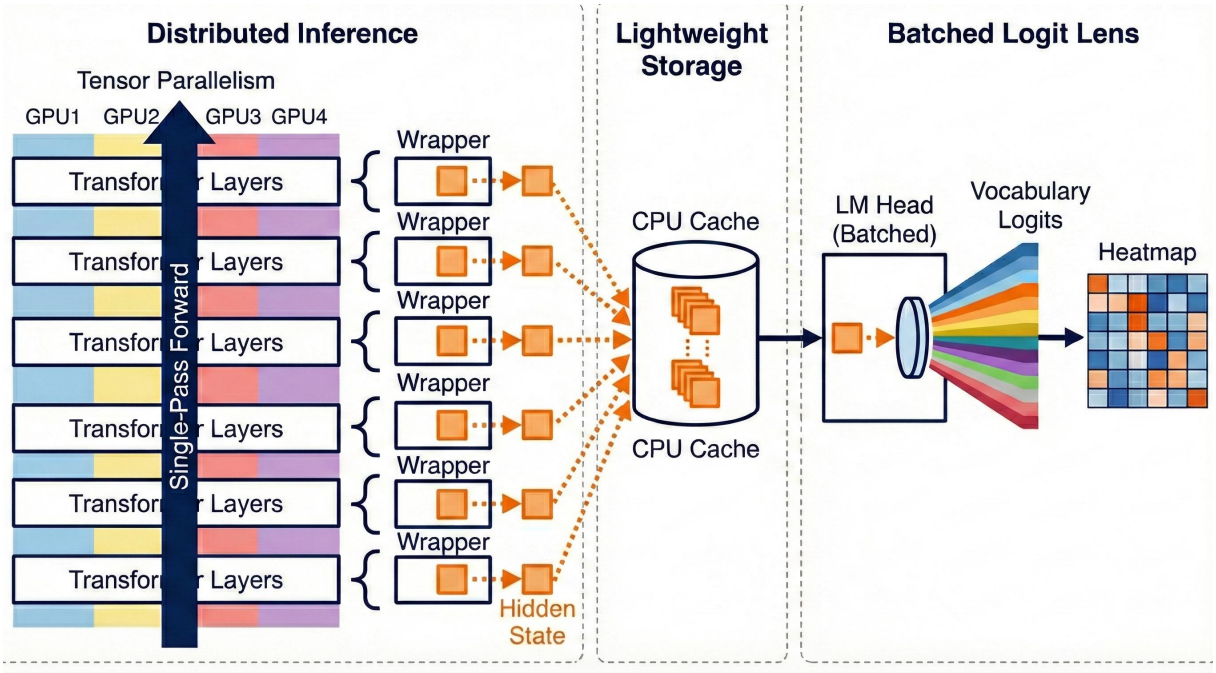

Figure 1: **Distributed single-pass logit-lens architecture.** Selected transformer layers are instrumented with lightweight wrappers that expose post-attention activations under tensor-parallel execution. Hidden states for the most recent token are recorded during a single KV-cached forward pass. After generation, all captured activations are decoded in batch via the final normalization layer and shared LM head, enabling scalable per-token, per-layer analysis without additional forward passes.

steering vectors across models and layers, showing that effectiveness varies substantially with architecture and injection depth. Empirical demonstrations on deployed models include (Gunton, 2024), who showed that mean-direction steering can effectively guide generative models toward improved reasoning behavior. Other work explores reducing interference: (Konen et al., 2024) introduced style vectors for controlled stylistic shifts, while (Stoehr et al., 2024) proposed sparse, layer-selective activation scaling to minimize side effects.

Open-source libraries such as `llm-steer` (Chirculescu, 2025) and `steering-vectors` (steering-vectors contributors, 2025) provide practical interfaces for constructing and applying steering vectors, though they do not yet address large-scale, tensor-parallel deployment.

2.3 Engineering and Systems for Interpretability at Scale

Several systems aim to support activation-level interpretability in large transformers, but most remain limited by model scale or execution constraints. Early pipelines for GPT-2 relied on per-layer projection or extensive activation logging, making them prohibitively expensive for larger models. More recent inference engines and tracing frameworks enable activation logging but do not support full residual-stream decoding or multi-layer steering during generation.

Crucially, prior systems do not jointly support *single-pass activation capture*, *deferred batched projection*, and *multi-layer activation intervention* while preserving tensor-parallel throughput and KV-cache semantics. As a result, interpretability and steering methods have remained largely incompatible with realistic multi-GPU inference for 10B+ parameter models.

3 System Architecture and Method

3.1 Architecture Overview

We operationalize the design principles introduced earlier single-pass activation capture, tensor-parallel execution, and deferred vocabulary projection into a three stage architecture illustrated in Fig. 1: (1) distributed model initialization with selective layer instrumentation, (2) single-pass autoregressive generation with activation capture, and (3) batched post-hoc decoding and serialization.

Stage 1: Distributed initialization and layer instrumentation. We initialize the tokenizer and

base language model under a tensor-parallel inference engine, partitioning weights across all available GPUs. A configurable subset of transformer layers is instrumented by replacing the original block with a *Block Output Wrapper*. Each wrapper exposes post-attention activations from three locations: (i) the attention mechanism output, (ii) the MLP output, and (iii) the block (residual-stream) output. The model's final normalization layer and shared LM head remain unmodified and are reused during post-hoc decoding.

Stage 2: Single-pass generation with activation capture. Generation proceeds via standard autoregressive decoding with key-value caching enabled. Because instrumentation occurs directly within the forward path, no additional forward passes are required. At each decoding step and for each wrapped layer, we record only the hidden-state slice corresponding to the most recently generated token.

Concretely, for a wrapped layer ℓ and activation type c, the recorded tensor has shape $[1, 1, d]$ (batch size 1, single token, hidden dimension d). These slices are appended during generation. Across a generation of length T, each wrapped layer therefore accumulates an activation trajectory of shape $[1, T, d]$ per activation type. All capture occurs within the same tensor-parallel, KV-cached forward pass as standard inference.

Stage 3: Batched decoding and serialization. After generation completes, the per-token activation slices are concatenated into full hidden-state trajectories. For each wrapped layer and activation type, we apply the model's final normalization followed by a batched projection through the shared LM head, yielding vocabulary logits for all tokens simultaneously. Rather than retaining full $|V|$-dimensional distributions, we extract only the top-k candidates and their probabilities per token. The resulting outputs are serialized in a JSON format, indexed by layer and activation type, and consumed by downstream visualization and analysis tools. This design ensures that memory usage scales with sequence length and hidden dimension, rather than vocabulary size.

Algorithm 1 Single-pass activation capture with batched logit-lens decoding

Require: Model $\mathcal{M}$, Tokenizer τ, Wrappers $\mathcal{L}$, Types $\mathcal{C}$
1: Initialize DeepSpeed inference with tensor parallelism
2: Instrument layers $\ell \in \mathcal{L}$ with Block Output Wrappers
3: **Phase 1: Generation & Capture**
4: Run autoregressive generation ($t = 1 \dots T$) with KV-cache:
5: Wrappers record slice $h \in \mathbb{R}^{1\times1\times d}$ at step t
6: **Phase 2: Post-hoc Decoding**
7: **for** $\ell \in \mathcal{L}, c \in \mathcal{C}$ **do**
8: Concatenate slices to form $H^{(\ell,c)} \in \mathbb{R}^{1\times T\times d}$
9: $Z \leftarrow \text{LMHead}(\text{LN}(H^{(\ell,c)}))$ ▷ Batched Projection
10: Extract top-k and Softmax(top-k)
11: **end for**

3.2 Single-pass Capture and Batched Decoding Algorithm

Memory complexity. Storing captured activations scales as:

$$\mathcal{O}(T \cdot d \cdot |\mathcal{L}| \cdot |\mathcal{C}|)\,.$$

For example, with $T = 1500$ tokens, $d = 8192$ vector embeddings size , $|\mathcal{L}| = 80$ transformer layers and C = number of activation types this corresponds to approximately 9.8×10^8 elements. Under bfloat16 precision (2 bytes per element), this is roughly 1.97 GB per activation type (or ~ 5.9 GB when storing attention, MLP, and block outputs), which remains tractable on modern multi-GPU systems.

3.3 Logit Lens Decoding

Our implementation of the logit lens operates as a post-hoc decoding step over the hidden-state trajectories captured during generation. By deferring projection into vocabulary space, we avoid repeated $|V|$-dimensional computation during autoregressive decoding.

Batched projection. Given a hidden state $\mathbf{h}_{\ell,t} \in \mathbb{R}^d$ from layer ℓ at token position t, we compute vocabulary logits as

$$\mathbf{z}_{\ell,t} = W_{\text{out}}\,\text{LN}(\mathbf{h}_{\ell,t}) + \mathbf{b}, \qquad (1)$$

where LN denotes the model's final normalization layer and W_{out} is the language model head of the model.

Top-k extraction. For each token position, we extract the top-k logits using an argsort operation. The value of k is specified by a user-controlled hyperparameter. Probabilities are computed by applying a softmax over the top-k logits only, yielding a conditional distribution over the selected candidates rather than a full-vocabulary normalization. This choice substantially reduces memory and compute overhead while preserving the relative ranking required for interpretability.

3.4 Activation Steering

Our instrumentation framework supports *activation-level steering* during autoregressive generation. Steering is compatible with DeepSpeed tensor parallelism and is applied post-LayerNorm, enabling controlled interventions without modifying model parameters or introducing additional forward passes.

Steering direction construction. Given a base sequence y^- and a target sequence y^+, we define a normalized steering direction

$$v = \frac{y^+ - y^-}{\|y^+ - y^-\|_2}. \tag{2}$$

Here, y^- and y^+ are hidden-state vectors extracted from the same layer and activation type under contrasting conditions (e.g., baseline vs. target behavior). The ℓ_2 normalization removes magnitude effects, isolating the direction of change. Steering vectors may be computed per layer or reused across layers.

Steering during generation. During controlled decoding, a scaled multiple of v is added to the attention output within the wrapped layer:

$$h' = h + \alpha v, \tag{2}$$

where α controls the strength of the intervention. Because steering is applied inside the tensor-parallel forward path, it preserves KV-cache reuse and the single-pass execution profile.

Stability considerations. Excessive steering strength may induce degeneration or loss of fluency. We mitigate these effects using magnitude clipping and layer-specific scaling. Across all evaluated models (LLaMA-3.1–8B/70B and Qwen-3–4B/14B/32B), this approach yields stable, monotonic behavioral shifts, demonstrating that the proposed architecture supports both scalable interpretability and real-time control at frontier model scale.

4 Experimental Setup

Hardware and software. All experiments were conducted on a single node equipped with $4\times$ NVIDIA RTX A6000 GPUs (48 GB each) and approximately 250 GB of system RAM. We use `DeepSpeed` for tensor-parallel inference and activation capture, together with `PyTorch` and the Hugging Face `Transformers` library. Unless otherwise stated, inference is performed in `bfloat16` precision with `tensor_parallel_size=4`.

Models. We evaluate five dense, decoder-only large language models spanning a wide range of scales and architectures: LLaMA 3.1–8B (Meta AI, 2024b), LLaMA 3.1–70B (Meta AI, 2024a), Qwen3–4B (Alibaba Cloud Qwen Team, 2025c), Qwen3–14B (Alibaba Cloud Qwen Team, 2025a), and Qwen3–32B (Alibaba Cloud Qwen Team, 2025b). All models were loaded from their public Hugging Face checkpoints and instrumented using identical post-LayerNorm wrappers. Minor adapter changes were required to account for model-specific normalization layouts and module naming.

Logit-lens protocol. We evaluate logit-lens interpretability using a fixed suite of general-knowledge and open-ended prompts (e.g., "Where is the oldest technical university located in America?"). For each prompt, we decode a budget of $B \in \{100, 300, 500, 700, 1500\}$ tokens for LLaMA 3.1–8B and $B \in \{500, 700, 1500, 2000, 2500\}$ tokens for LLaMA 3.1–70B.

During decoding, we enable hooks to capture *all intermediate activations* at every layer, including attention-mechanism outputs, MLP outputs, residual streams, and post-LayerNorm block outputs. Activations are recorded only for the final token at each decoding step and stored as compact per-token hidden-state slices. After decoding completes, all captured states are projected in batch through the shared LM head to produce layer-wise logit-lens predictions.

For each configuration, we report end-to-end

Tokens	LL4L–TPtime (s)	SP–TPtime (s)	Speedup (×)	Baseline tok/s	Ours tok/s
100	166.0 ± 4.8	4.0 ± 0.3	41.5	0.60 ± 0.02	25.0 ± 1.8
300	498.0 ± 11.2	15.0 ± 0.9	33.2	0.60 ± 0.01	20.0 ± 1.2
500	833.0 ± 19.5	20.0 ± 1.1	41.7	0.60 ± 0.01	25.0 ± 1.4

Table 1: Baseline (LL4L–TP) vs. our method (SP–TP) logit-lens runtime comparison on LLaMA-3.1–8B. Wall-clock time and throughput are reported as mean $\pm$ std over $N = 3$ runs. Speedup is computed as the ratio of (LL4L–TP) time to (SP–TP) time. Both methods use identical multi-GPU tensor-parallel inference (4× RTX A6000, bfloat16). The baseline performs per-layer re-forwarding, while ours method uses single-pass capture with deferred batched projection.

wall-clock runtime and throughput (tokens/s), averaged over prompts. Timing measurements exclude model loading and include a short warm-up phase.

Baselines. As no official throughput benchmarks exist for `LogitLens4LLMs` (Wang, 2025), we reimplemented and evaluated it as a baseline under controlled conditions. Specifically, we ran the original single-GPU implementation on LLaMA 3.1–8B using a single RTX A6000 (48 GB), decoding $B \in \{100, 300, 500, 700\}$ tokens. For each run, we report end-to-end runtime and compute throughput as tokens/s $= B/\text{Duration}$.

The original `LogitLens4LLMs` (Wang, 2025) implementation could not be executed on multi-GPU setups or on larger models (e.g., LLaMA 3.1–70B, Qwen3–32B) due to out-of-memory failures when decoding long sequences. Accordingly, we treat LLaMA 3.1–8B as the largest reproducible single-GPU baseline and use it as a reference point for evaluating the scalability of our distributed implementation (Table 1).

Steering protocol. We evaluate activation steering on three behavioral datasets introduced by (Perez et al., 2022): Corrigible–Neutral HHH, Self-Aware LM, and Power-Seeking LLM. Each prompt is formatted as a multiple-choice question terminating in a single-token label (A/B).

Steering vectors are computed at the label token position by subtracting matched base (y^-) and target (y^+) activations, followed by ℓ_2 normalization. Vectors are injected post-LayerNorm at a specified layer L during generation. We ablate injection layer and injection point (post-LayerNorm vs. pre-LayerNorm) and compare against last-token steering. Evaluation follows (Tan et al., 2024), reporting steerability slope and mean Δ change across steering multipliers α.

Model	Params	Mean tok/s	Std
LLaMA-3.1–8B	8B	23.3	2.6
LLaMA-3.1–70B	70B	21.0	1.2
Qwen3–4B	4B	28.5	1.9
Qwen3–14B	14B	21.2	1.4
Qwen3–32B	32B	17.0	1.1

Table 2: **Cross-model throughput summary.** comparison of average decoding throughput for SP–TP. Mean tok/s is aggregated across the token budgets used for each model; Std is across runs ($N = 3$). (Replace dummy values with measured results.)

5 Results

Evaluation protocol. Unless otherwise stated, all runtime and steering experiments are repeated $N = 3$ times with identical prompts and decoding budgets. We report mean $\pm$ standard deviation for throughput metrics and regression-based statistics for steering analyses. Timing excludes model loading and includes a short warm-up phase .

5.1 Runtime and Throughput Scaling

LLaMA 3.1–8B (baseline comparison). Table 1 reports a direct comparison between the LogitLens4LLMs tensor-parallel baseline (LL4L–TP) and our single-pass, deferred-projection method (SP–TP) on LLaMA 3.1–8B. Across token budgets of 100–500 tokens, SP–TP achieves a $41\times$ reduction in wall-clock time while sustaining 20–25 tokens/s. In contrast, the baseline remains bottlenecked at approximately 0.6 tokens/s due to per-layer re-forwarding. All values are reported as mean $\pm$ standard deviation over $N = 3$ runs.

Runtime scaling across models. To evaluate how runtime scales with model size and token budget, Table 3 reports end-to-end runtime and throughput for SP–TP across five models spanning two architectures (LLaMA 3.1 and Qwen3) and parameter scales from 4B to 70B. Despite substantial differences in model size and hidden dimension,

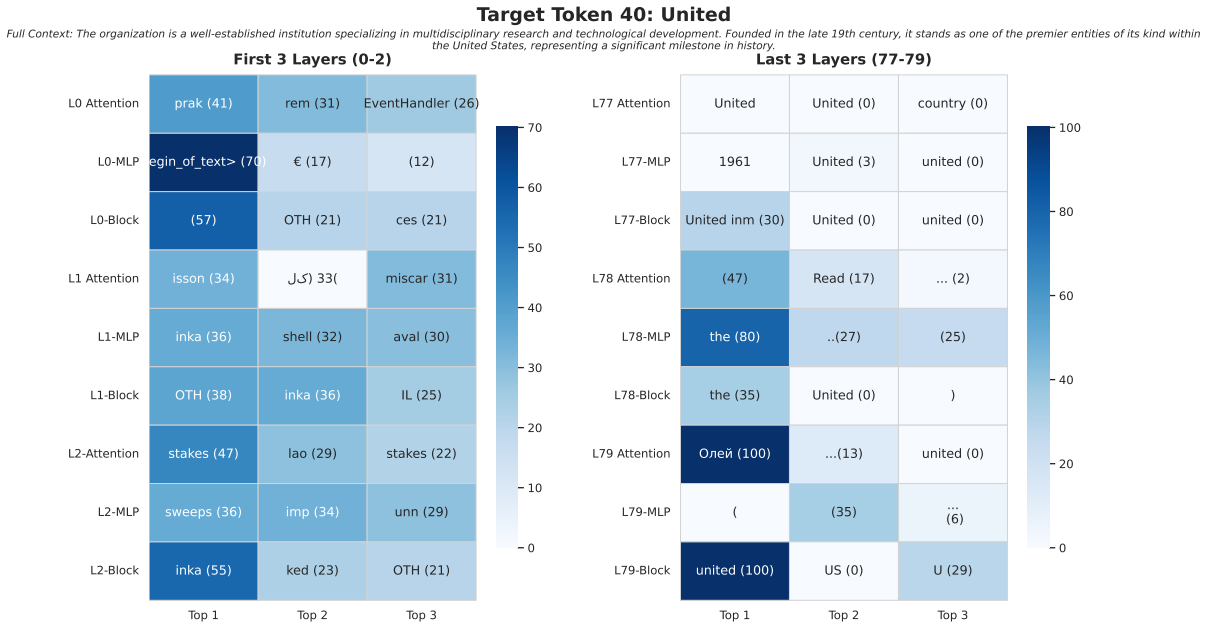

Figure 2: Logit lens heatmap for the token “university” across all 80 layers of the LLaMA 3.1–70B model. For each layer, three types of activations are visualized *attention*, *MLP*, and *block output* representing the intermediate computations captured after the post-LayerNorm stage. Each layer displays the model’s top-$k = 3$ most probable tokens, and the color shade indicates their probability: darker shades correspond to higher token probabilities within that layer. Due to space constraints, only the first three and last three layers are shown here. Some cells display non-English or alphanumeric characters (e.g., “\u06\u0644”), which occur when the model emits tokens from other scripts or symbols not supported by the visualization font.

throughput remains stable in the 20–100 tokens/s range, indicating that performance is governed primarily by tensor-parallel inference and batched LM-head projection.

LLaMA 3.1–70B scalability. Within Table 3, we observe that SP–TP scales reliably to LLaMA 3.1–70B with decode budgets up to 2,500 tokens. A 1,500-token run completes in 75 ± 3.8 seconds while maintaining approximately 20 tokens/s throughput. Baseline methods could not be executed at this scale due to out-of-memory failures, consistent with prior reports.

Beyond throughput, this scale allows for granular inspection of the model’s internal reasoning. As illustrated in Figure 2, we visualize the evolution of the target token representation across the 80-layer architecture. While early layers (0–2) contain high-entropy activations and syntactic markers, the final layers (77–79) demonstrate the ”logit lens” effect, where the model’s internal state converges on the target token “United” with near-total probability.

5.2 Cross-Model Throughput Consistency

Table 2 provides a collapsed, reviewer-friendly summary of mean throughput across all evaluated models. Across both LLaMA 3.1 and Qwen3 families, throughput varies by less than $1.7\times$ despite nearly an order-of-magnitude change in parameter count. This consistency suggests that logit-lens performance under our method is largely architecture-agnostic once tensor-parallel inference is fixed.

5.3 Steering Dose Response and Layer Localization

For each prompt, we compute the mean label-token propensity as a function of steering multiplier $\alpha \in [-1.5, 1.5]$. A linear regression is fit per prompt, and slopes are averaged across prompts. Aggregated steering statistics are reported in Table 4, including mean slope, coefficient of determination (R^2), and statistical significance. Mid-layer interventions exhibit strong, monotonic dose–response behavior as illustrated in figure 3 at layer $L = 35$ in LLaMA 3.1–70B, steering achieves a mean slope of 0.70 with $R^2 = 0.85$. In contrast, earlier layers show substantially weaker effects, with slopes near zero and non-significant p-values.

Layer efficiency. Achieving a $+0.3$ increase in propensity requires a multiplier of approximately $0.35\times$ at layer 35, compared to 4–$6.5\times$ at deeper

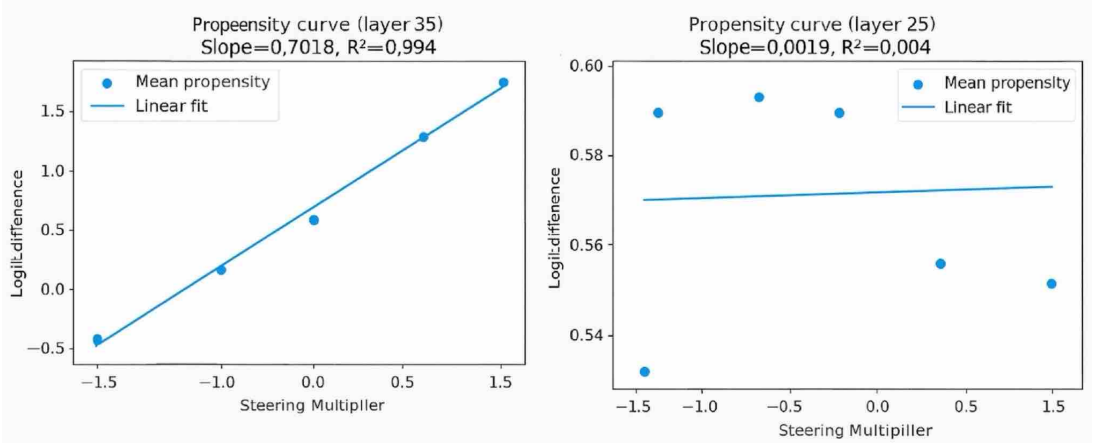

Figure 3: Steering-vector propensity analysis on LLaMA 3.1–70B. Mean propensity vs. steering multiplier α for selected layers. Steerability peaks at mid layer (L=35) and is weaker at early layers (L=25), consistent with directional sensitivity patterns reported by Tan et al. (2024).

Model	Params	Tokens	Time (s) (mean ± std, N=3)	tok/s (mean ± std)
LLaMA-3.1–8B	8B	100	4.0 ± 0.3	25.0 ± 1.8
LLaMA-3.1–8B	8B	300	15.0 ± 0.9	20.0 ± 1.2
LLaMA-3.1–8B	8B	500	20.0 ± 1.1	25.0 ± 1.4
LLaMA-3.1–70B	70B	500	25.0 ± 1.5	20.0 ± 1.1
LLaMA-3.1–70B	70B	1500	61.3 ± 3.8	20.0 ± 1.0
LLaMA-3.1–70B	70B	2500	125.0 ± 6.2	20.0 ± 1.0
Qwen3–4B	4B	500	18.0 ± 1.0	27.8 ± 1.6
Qwen3–14B	14B	500	24.0 ± 1.4	20.8 ± 1.2
Qwen3–32B	32B	500	30.0 ± 1.9	16.7 ± 1.1

Table 3: Runtime scaling across models. Wall-clock time and throughput for SP–TP over varying token budgets. Reported values are mean ± std over $N = 3$ runs on identical hardware (4× RTX A6000).

Model	Layer	Mean slope	p-value
LLaMA-3.1–8B	35	0.68	0.004
LLaMA-3.1–8B	62	0.07	0.180
LLaMA-3.1–70B	35	0.72	0.002
Qwen3–14B	35	0.55	0.011
Qwen3–32B	35	0.49	0.019

Table 4: Steering statistics for each (model, dataset, layer), we fit a per-prompt linear regression of the steering metric versus multiplier $m \in [-1.5, 1.5]$, then report the mean slope ± std across prompts. The p-value is computed via a paired test comparing the metric at $m = -1.5$ vs. $m = +1.5$.

layers. This gap indicates a localized leverage region in mid-depth transformer blocks, consistent with prior steering analyses (Tan et al., 2024).

5.4 Failure Modes and Saturation

At large multipliers ($|\alpha| > 1.5$), steering effects saturate and may induce mild fluency degradation; these regimes are excluded from quantitative analysis. Early-layer injections ($L < 20$) produce near-zero slopes ($R^2 < 0.01$), suggesting insufficient semantic separation at those depths.

6 Conclusion

We close a practical gap: token-level interpretability and targeted interventions on 4B to 70B LLMs without extra passes or slowdowns. Our single-pass, tensor-parallel instrumentation with deferred LM-head projection, plus residual-stream steering computed at the label token and injected post-LN (pre-MLP; empirically better than pre-LN), sustains 0.01–0.05 s/token (20–100 tok/s) and completes a 1,500-token run in ≈60 s, with mean steerability slope 0.702. Compared to prior single-GPU baselines, it is up to **41**× faster while logging per-token/per-layer signals and supporting scheduled injections. Our results expand the avenues towards building unified interpretability and steering systems on multi-GPU LMs.

Limitations

Our evaluation focuses on decoder-only transformer language models, and extending the framework to encoder decoder or non-transformer architectures remains future work. The proposed framework operates entirely at inference time and does not address training-time interpretability or representation learning dynamics. Steering directions are constructed from contrastive sequence pairs, which we find effective in practice, though alternative formulations (e.g., subspace-based or nonlinear directions) are not explored. Finally, experiments are conducted in multi-GPU tensor-parallel environments, and performance characteristics may vary under different hardware or parallelization regimes.

References

Alibaba Cloud Qwen Team. 2025a. Qwen3-14b — model card. Hugging Face. Accessed 2025-08-23.

Alibaba Cloud Qwen Team. 2025b. Qwen3-32b — model card. Hugging Face. Accessed 2025-08-23.

Alibaba Cloud Qwen Team. 2025c. Qwen3-4b — model card. Hugging Face. Accessed 2025-08-23.

Nora Belrose, Igor Ostrovsky, Lev McKinney, Zach Furman, Logan Smith, Danny Halawi, Stella Biderman, and Jacob Steinhardt. 2023. Eliciting latent predictions from transformers with the tuned lens. *arXiv preprint arXiv:2303.08112*. V5.

Mihai Chirculescu. 2025. llm_steer: Activation-engineering steering vectors for large language models. GitHub repository. Accessed 2025-12-01.

Matthew Gunton. 2024. Using vector steering to improve model guidance. Medium. Accessed 2025-12-01.

Kai Konen, Sophie Jentzsch, Diaoulé Diallo, Peer Schütt, Oliver Bensch, Roxanne El Baff, Dominik Opitz, and Tobias Hecking. 2024. Style vectors for steering generative large language models. *arXiv preprint arXiv:2402.01618*.

Meta AI. 2024a. Llama 3.1 70b — model card. Hugging Face. Accessed 2025-08-23.

Meta AI. 2024b. Llama 3.1 8b — model card. Hugging Face. Accessed 2025-08-23.

Nostalgebraist. 2020. Interpreting GPT: The logit lens. LessWrong. Accessed 2025-12-01.

Ethan Perez, Sam Ringer, Kamilė Lukošiūtė, and 1 others. 2022. Discovering language model behaviors with model-written evaluations. arXiv preprint arXiv:2212.09251.

steering-vectors contributors. 2025. Steering vectors. GitHub repository. Accessed 2025-12-01.

Niklas Stoehr and 1 others. 2024. Activation scaling for controlling large language models. arXiv preprint.

Daniel Tan, David Chanin, Aengus Lynch, Dimitrios Kanoulas, Brooks Paige, Adria Garriga-Alonso, and Robert Kirk. 2024. Analyzing the generalization and reliability of steering vectors. *Preprint*, arXiv:2407.12404.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J. Vazquez, Ulisse Mini, and Monte MacDiarmid. 2023. Steering language models with activation engineering. *arXiv preprint arXiv:2308.10248*.

Zhenyu Wang. 2025. LogitLens4LLMs: A logit lens toolkit for modern large language models. Computer software. Accessed 2025-12-01.