LayerLens: Watch LLMs think Layer by Layer

by

Yash Rathi

June 1, 2026

A Project submitted to the
faculty of the Graduate School of
the University at Buffalo, The State University of New York
in partial fulfillment of the requirements for the
degree of

Master of Science
Department of Computer Science

Acknowledgments

I would like to thank my advisor, Dr. Kenneth W. Regan for his guidance and support throughout this project. His feedback shaped both the direction of the research and the rigor with which the findings were evaluated. I would also like to thank Dr. Cassandra Jacobs for their valuable input and encouragement at key stages of this work.

Abstract

Language model confidence is typically measured by softmax probability at the final output layer, a signal that reflects only the model's terminal state and discards all information about how a prediction developed across intermediate layers. This makes softmax a poorly calibrated uncertainty measure. No lightweight method exists to profile how a model distributes linguistic processing across depth for different word types without training additional probe classifiers that require labeled data and model-specific compute.

This project introduces LayerLens, a framework that applies the LogitLens technique to track rank trajectories for every token across every transformer layer. Each intermediate hidden state is projected through the model's own output head, requiring no additional parameters or training data. Experiments are conducted on a 2,500-sentence corpus spanning five domains using two model architectures. Three trajectory features (monotonicity, entry speed, and persistence) form an uncertainty score that is evaluated against softmax probability using area under the ROC curve (AUROC).

LayerLens produces three main findings. First, parts of speech saturate at statistically distinct layers, with a gap of 5.9 layers between the easiest and hardest word types. Second, tokens that drop in rank only in the final quarter of processing are incorrect 86.1 percent of the time under soft correctness scoring, a failure mode that final-layer softmax cannot detect. Third, trajectory-based uncertainty achieves an AUROC of 0.8151 compared to 0.7783 for softmax, a consistent improvement across four natural language domains and 32,453 tokens.

Table of Contents

List of Tables

List of Figures

Introduction

Transformer language models process input as a sequence of discrete layers. At each layer, the model updates its internal representation of every token through self-attention and feed-forward operations, producing progressively refined hidden states (Vaswani et al., 2017). By the final layer, these hidden states are projected through the model's output head to produce a probability distribution over the vocabulary, and the token with the highest probability is selected as the model's prediction. This final-layer view is the standard interface through which users and researchers interact with language model outputs.

The dominant measure of model confidence[1] is the softmax probability assigned to the top-predicted token at this final layer. However, softmax probability is a well-documented poor calibration signal: a model can assign 90% confidence to a prediction that is factually wrong, and high confidence does not reliably correlate with correctness (Guo et al., 2017). A deeper reason for this miscalibration is that softmax observes only the model's terminal state. A token whose rank was highly unstable through most of the computation but happened to land well at the last layer looks identical to one that was

[1] This use of "confidence" is consistent with current LLM parlance, but we also adopt a wider view of what the correctness of a projected probability p_w, for the next word w at a juncture of a given text t, means: Over aggregates of texts and junctures where the (top) model probability p' (approximately) equals p_w, the proportion p* in which the associated w' is the correct ground-truth word equals p_w. Then the "internal confidence" would be a number d, possibly also generated by the model, such that over sets S of such aggregates, the measured p* satisfies p_w-d <= p* <= p_w+d in, say, 95% of the aggregates in S. This notion of predictive accuracy applies to the p_w given to any vocabulary token, not just the model's top token. It is captured more nimbly by the *Brier score* and related tools, which may undergird future work---in which we say "model projected probability" rather than "model confidence".

confidently resolved from early layers onward. The computational history that produced the final output is discarded entirely.

The conventional approach for studying what happens inside transformer layers is the probe classifier (Tenney et al., 2019; Belinkov, 2022). A probe is a small supervised classifier trained on the hidden states of a target layer to predict a linguistic property such as part of speech (POS), syntactic role, or semantic category. Probing studies have established that transformers encode increasingly abstract linguistic structure across depth, with surface-level features emerging in early layers and complex semantic or relational properties appearing in deeper ones. However, probing classifiers require labeled training data, plus dedicated compute for each layer and each linguistic property. They must be retrained when the model changes. These requirements make probing impractical as a routine tool for model auditing or rapid analysis of new models.

An alternative approach was introduced by Nostalgebraist (2020) under the generic name “LogitLens”. Rather than training a separate classifier, the LogitLens projects each intermediate hidden state directly through the model's own output head, reading off what the model "would have predicted" at that layer with zero additional parameters. This makes every intermediate state interpretable in the same vocabulary space as the final output. Belrose et al. (2023) extended this idea with the “TunedLens”, which adds a learned affine correction at each layer to improve the quality of intermediate projections. Both methods make intermediate model representations accessible, but neither has been applied systematically to characterize linguistic

processing patterns across word types or to construct a trajectory-based uncertainty signal.

This project introduces “LayerLens”, a framework that builds on the LogitLens in its original, unmodified form (referred to as the vanilla LogitLens) to address three problems that remain open. First, it provides a zero-training-cost linguistic profile of any transformer model, measuring how different parts of speech distribute their processing across depth. Second, it constructs an uncertainty score from the shape of each token's rank trajectory across layers and shows that this trajectory-based signal predicts correctness better than final-layer softmax probability. Third, it quantifies where in the network a token's prediction has effectively concluded, making the structural mismatch between word complexity and compute allocation measurable and visible.

System Design and Methodology

System Overview

LayerLens operates in three stages. First, it runs a logit lens inference pass on a pre-trained transformer model to extract a rank trajectory for every token across all layers. Second, it applies feature extraction and classification to characterize the shape of each trajectory, producing per-token uncertainty scores and linguistic profiles. Third, it evaluates those scores against ground-truth correctness and compares processing patterns across model architectures. Figure 1 shows the overall system architecture.

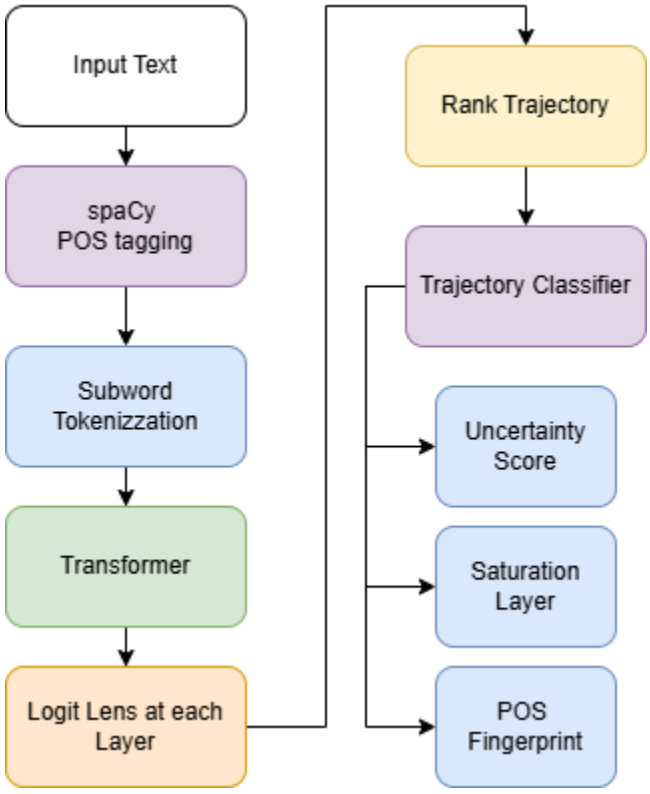

Figure 1 LayerLens system architecture

LayerLens Inference Pipeline

At every layer ℓ of a transformer with L total layers, the model produces a hidden state h_ℓ for each input token. In standard inference, only the final hidden state h_L is projected through the output head to generate predictions. LayerLens applies this projection at every intermediate layer: for each hidden state h_ℓ, it applies the model's own final layer normalization followed by the output embedding matrix to obtain a logit vector of size V (the vocabulary size). The rank of the correct next token within this logit vector is recorded at every layer, producing a rank trajectory of length L per token.

Applying the final layer normalization before projection is critical. Without it, intermediate hidden states are not scaled appropriately for the output head, and logit values at early layers are statistically unreliable. For Qwen3-0.6B, all aggregate statistics (mean saturation layers, trajectory type distributions, AUROC evaluations, and early exit results) are restricted to layers 10 through 28. Saturation detection searches all layers from 1 to L, but any token whose *saturation layer* (as defined on page 7) falls below 10 is excluded from the aggregate statistics reported in Tables 4 and 6. For GPT-2, which has only 13 layers total, applying a layer-10 cutoff would leave only four layers for analysis, making per-POS aggregate statistics unreliable at the opposite extreme. GPT-2 aggregate statistics therefore use all layers; this difference is noted in the cross-model comparison, which avoids the issue by normalizing layer position to a fraction

between 0 and 1 before comparing the two architectures. Early layers are included in all visualizations but are rendered with reduced opacity to signal their unreliability.

Figure 2 shows the logit jump heatmap produced by the inference pipeline: the mean change in logit value per part of speech at each layer. The green band visible at layers 12 to 18 for content words (nouns, verbs, and adjectives that carry meaning) and the earlier band at layers 5 to 10 for function words (grammatical words such as determiners, conjunctions, and prepositions that carry structure rather than meaning) illustrate POS-layer specialization directly from the raw inference output.

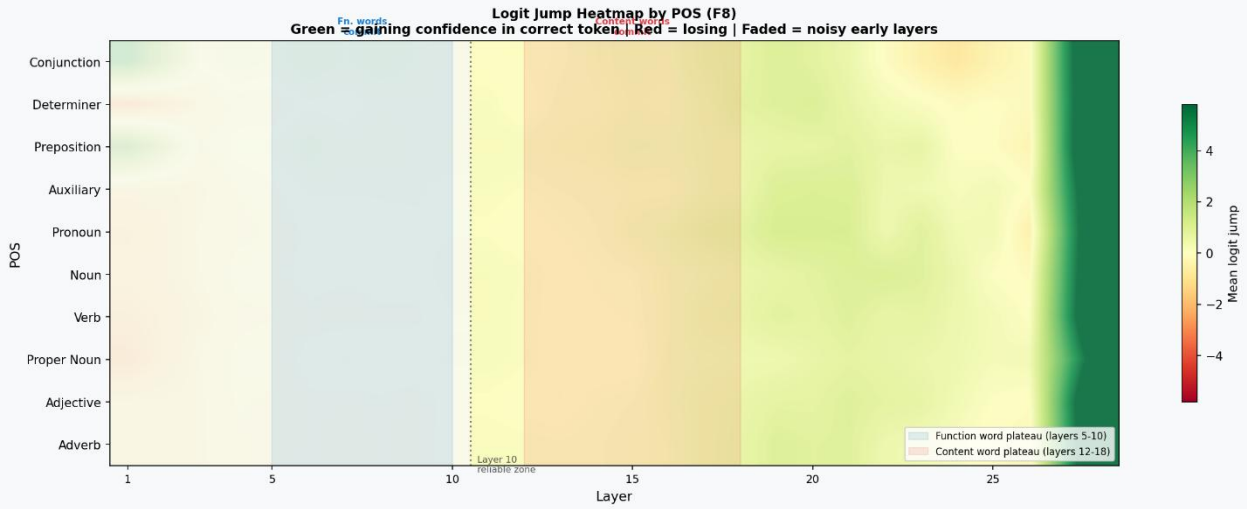

Figure 2 Logit jump heatmap, Qwen3-0.6B. Data from Table 3

Rank Computation and Normalization

The raw rank of a token at layer L is the number of vocabulary tokens with a higher logit value, plus one. This ranges from 1 (the model's top prediction) to V (last in the vocabulary). To enable direct comparison across models with different vocabulary sizes, raw rank is converted to a percentile rank by dividing by V, yielding a value between 0

and 1. This normalization means that a rank of 0.01 carries the same meaning regardless of whether the model has 50,000 or 150,000 tokens in its vocabulary.

For words that span multiple subword tokens, LayerLens uses the maximum percentile rank across all subwords at each layer as the bottleneck value. This reflects the principle that a multi-part word is only resolved when its hardest subword piece is resolved. For example, a word like "bioluminescence" tokenized into four subwords is considered saturated only when all four subwords reach stable, high-confidence ranks.

Saturation Detection and Trajectory Classification

The *saturation layer* for a token is defined as the first layer at which the token's rank remains exactly stable across five consecutive layers. Tokens that never achieve this stability are assigned the final layer index and are included in all statistics.

Each token's rank trajectory is then classified into one of three types based on its shape. The first type, **Smooth Early**, describes a token whose rank increases steadily across layers and enters the top 100 of the vocabulary before the midpoint of the network. This indicates confident, early processing where the model commits to the correct token well before the final layer. The second type, **Late Rise**, describes a token that maintains a high rank throughout most of the forward pass and gains sharply only in the final 25 percent of layers. This pattern superficially resembles convergence but is associated with very high error rates, as the model appears to arrive at an answer at the last moment rather than through sustained confidence. The third type, **Flat or Erratic**,

groups two surface-distinct patterns that share the same diagnostic implication. A flat trajectory is one in which the token's rank stays stable at a consistently low position throughout all layers: the model holds a confident but wrong candidate and does not revise it. An erratic trajectory is one in which rank oscillates across layers without directional improvement: the model cycles through candidates without committing to any. In both cases the rank never enters the top 100 cleanly at any layer. The two are grouped into a single class because both reflect genuine confusion that persists across the full depth of processing, and both produce indistinguishable values on all three trajectory features.

These three types capture qualitatively different failure modes. Late Rise and Flat or Erratic tokens are both incorrect at high rates, but for different reasons: Late Rise reflects deep relational processing that almost converged, while Flat or Erratic reflects a prediction the model was never close to making.

Uncertainty Score Construction

Three features are extracted from each rank trajectory and normalized by L to make them architecture-agnostic across models of different depths.

Table 1 Trajectory features and formulas

Feature	Formula	What It Captures
Monotonicity (M)	Fraction of steps where rank increased	How smoothly rank increased toward the top
Entry Speed (E)	(L - entry_layer) / L	How early the token entered the top 100
Persistence (P)	Layers in top 10 / L	How long the token stayed near the top

The uncertainty score is defined as:

$$\mathbf{U(t) = 1 - (\alpha \cdot M + \beta \cdot E + \gamma \cdot P)}$$

Weights α, β, and γ are learned via logistic regression on exact-match correctness labels from the full corpus. An equal-weight baseline (α = β = γ = 1/3) is also reported as an interpretable reference. The learned weights place the majority of influence on persistence (0.861), which reflects how consistently the model committed to the correct token in the final layers.

Soft Correctness Scoring

Exact-match correctness (whether the model's top-1 prediction matches the ground-truth next token) is a strict criterion that penalizes semantically valid predictions. To account for this, LayerLens applies a two-stage soft correctness check.

Stage 1 uses WordNet: for whole-word predictions, the prediction is marked correct if it appears in the WordNet synonym set of the ground-truth token. This handles clean natural language synonym cases cleanly.

Stage 2 uses calibrated embedding cosine similarity as a fallback for tokens not covered by WordNet. The similarity threshold is set as the midpoint between the median synonym similarity and the median antonym similarity measured on 100 calibration pairs from the model's own input embedding space. For Qwen3-0.6B, this threshold was 0.1883.

One known limitation of Stage 2 is that antonym pairs in static input embeddings tend to be more similar than synonym pairs due to shared distributional context. This means the semantic stage introduces some noise. Results from Stage 1 (WordNet) are treated as reliable; Stage 2 results are reported separately and are not used in headline claims about verb correctness.

Experimental Setup

Experiments are conducted on two decoder-only model architectures. Decoder-only models generate text by predicting one token at a time, left to right, using only the tokens that came before as context. This is the architecture used by the GPT family of models and most modern large language models. Two models are tested to assess whether findings hold beyond a single architecture.

Table 2 Transformer models used in all experiments

Model	HuggingFace ID	Layers	Vocabulary Size
Qwen3-0.6B	Qwen/Qwen3-0.6B	28	151,936
GPT-2	openai-community/gpt2	13	50,257

The evaluation corpus consists of 2,500 sentences drawn from five domains, with 500 sentences per domain sampled with a fixed random seed for reproducibility.

Table 3 Corpus composition by domain

Domain	Source Dataset	Sentences	Tokens
News	ag_news	500	9,199
Scientific	pubmed_qa	500	8,258
Creative	TinyStories	500	5,700
Social	tweet_eval	500	9,296
Code[2]	code_search_net	500	3,870

The code domain is analyzed separately from the four natural language domains, as part-of-speech tags are less meaningful for code identifiers and operators. All AUROC evaluations are conducted on the 32,453 tokens from the four natural language domains.

[2] Tokens from the Code domain are listed for completeness but are completely excluded from all reported accuracy and AUROC metrics in the present report. This data is isolated because standard part-of-speech tags are not meaningful for programming identifiers and operators.

Early Exit Simulation

A key question raised by saturation detection is whether the layers a model runs after a token has saturated are actually doing anything useful. The early exit simulation tests this directly. For each token, once its saturation layer S is identified, LayerLens freezes the token's hidden state at layer S and uses that frozen state to generate the final prediction, skipping all subsequent layers for that token. It then compares this early prediction to the prediction produced by full inference. If the two agree, the layers after S contributed nothing to that token's outcome.

This simulation is observational in scope. It measures whether a token's own prediction changes, but does not model the effect of a frozen hidden state on other tokens that attend to it in later layers. Results are reported as agreement rate (how often the early exit prediction matches full inference) and compute saved (the fraction of layers skipped relative to the full depth).

Cross-Model Comparison

To compare linguistic processing patterns across architectures with different depths, LayerLens constructs a fingerprint matrix for each model: a matrix of shape (Number of POS tags x Number of layer bins) where layer position is normalized to a fraction between 0 and 1. This normalization allows models with 13 layers and 28 layers to be compared on the same scale.

Two similarity metrics are computed between fingerprint matrices. Cosine similarity treats the matrix as a flat vector and measures overall shape agreement. Dynamic Time Warping (DTW) is applied per-POS row and measures whether the same processing pattern occurs at a different pace across the two architectures. A low DTW distance with moderate cosine similarity would indicate that both models run the same linguistic pipeline but at different speeds, which is a stronger universality claim than shape similarity alone.

Results

POS-Layer Specialization

Parts of speech saturate at statistically distinct layers across the full corpus (Kruskal-Wallis H = 2751.62, p < 0.001). Conjunctions reach a stable prediction at layer 21.9 on average, while proper nouns require until layer 27.9, a gap of 5.9 layers. Table 4 summarizes the saturation layers for key POS categories under the Qwen3-0.6B model, restricted to layers 10 through 28, and Figure 3 shows the full distribution.

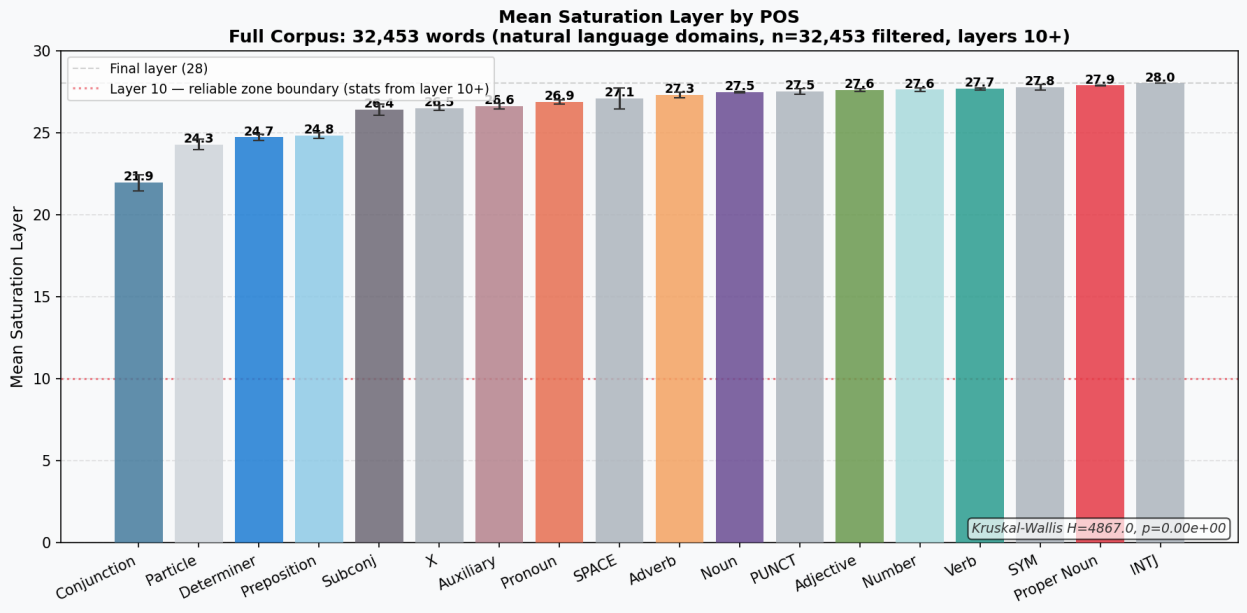

Figure 3 Mean saturation layer per part of speech, Qwen3-0.6B, full corpus.

Table 4 Mean saturation layer for selected parts of speech, full corpus for Qwen3

Part of Speech	Mean Saturation Layer	n
Conjunction	22	950
Determiner	25	2246
Preposition	25	3437
Auxiliary	27	1433
Noun	27	6361
Verb	27	3673
Proper Noun	28	4438

The ordering aligns with the linguistic distinction between function words and content words. Function words are constrained almost entirely by syntactic position and saturate early; content words depend on semantic context that only emerges in deeper layers. Never-correct rates (the fraction of tokens whose top-1 prediction never matched the actual next word at any layer) confirm this gap: verbs are never-correct 81.4 percent of the time, while determiners are never-correct only 32.6 percent of the time (Figure 4). Even after applying WordNet synonym scoring, 78.5 percent of verb tokens remain never-correct, suggesting the issue reflects genuine semantic ambiguity rather than a strict evaluation criterion. The POS saturation ordering holds across all four natural language domains (Spearman rho: 0.735 to 0.975) and across short, medium, and long sentence length buckets (rho ≥ 0.90 in every bucket), confirming that the finding is not domain-specific or an artifact of sentence length.

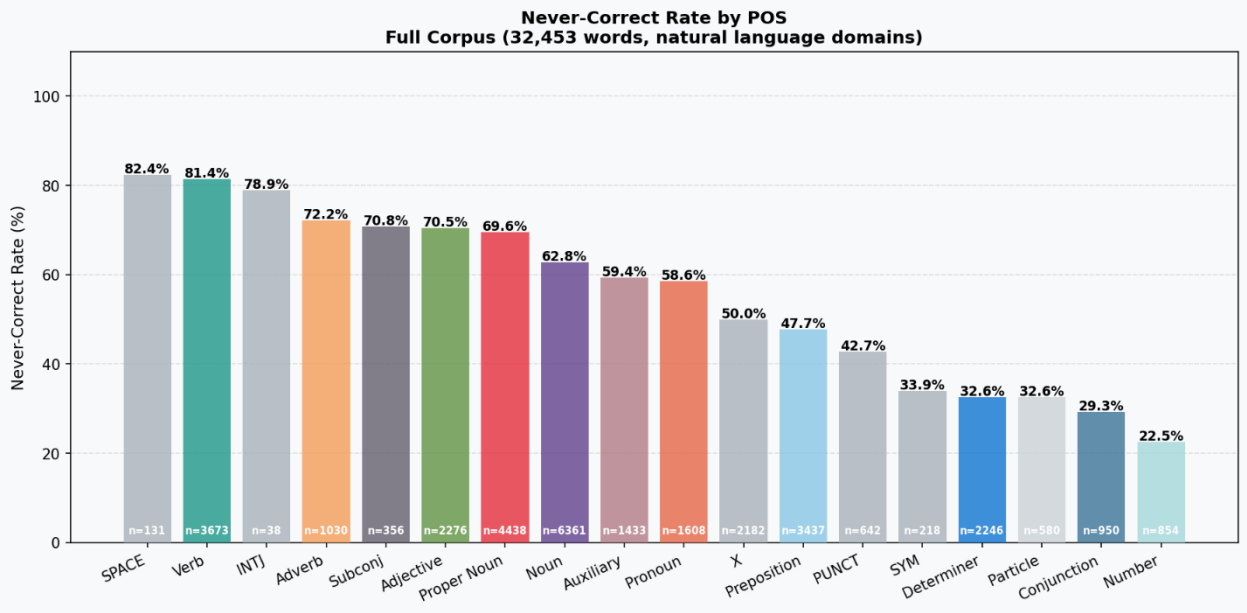

Figure 4 Never-correct rate per part of speech, full corpus.

Trajectory Classification and Uncertainty Scoring

Each token is classified into one of three trajectory types based on the shape of its rank curve. Table 5 reports the distribution and correctness rates across the full corpus.

Table 5 Trajectory type distribution and correctness rates, full corpus

Type	Tokens	Correct Rate*	Mean Softmax
Smooth Early	6308	47.4%	0.166
Flat or Erratic	23998	41.5%	0.142
Late Rise	6017	27.8%	0.035

*The "Correct Rate" in Table 5 (27.8% for Late Rise) measures strict exact matches at any intermediate layer across the entire 28-layer trajectory. Conversely, the

86.1% headline error rate refers strictly to the final layer's performance evaluated under soft correctness scoring (WordNet/embeddings). Because many Late Rise tokens briefly hit rank 1 during intermediate processing before destabilizing at the end, their any-layer exact match rate is naturally higher than their terminal success rate.

The Late Rise type is the central finding of this analysis. These tokens appear to converge at the last moment, yet they are wrong 86.1 percent of the time under soft correctness scoring. Their mean final softmax probability (0.0352) is near zero, but softmax cannot distinguish them from Smooth Early tokens at the final layer because it has no memory of the trajectory that preceded the output.

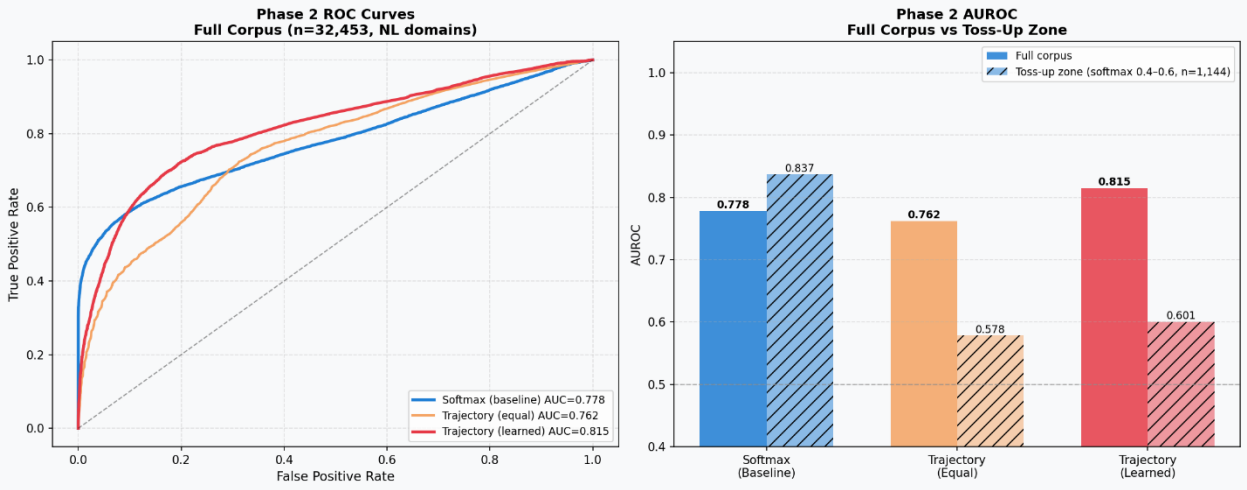

Figure 5 AUROC comparison: trajectory score vs. softmax, full corpus.

In the right panel of Figure 5, the toss-up zone bars show softmax at 0.837, trajectory equal-weight at 0.578, and trajectory learned at 0.601.

Early Exit Simulation

Stopping at the saturation layer and comparing the resulting prediction to full inference yields an overall agreement rate of 95.7 percent, meaning the layers after saturation change the outcome only 4.3 percent of the time. The effect is concentrated in function words: freezing all conjunctions at their average saturation layer saves 21.2 percent of compute while agreeing with full inference 84.6 percent of the time. Proper nouns save nothing; the model genuinely needs all 28 layers to process them. Table 6 gives the full breakdown by POS category for Qwen3-0.6B, restricted to layers 10 through 28.

Table 6 Early exit simulation results by part of speech for Qwen3

Part of Speech	Mean Saturation Layer	Agreement Rate	Compute Saved
Conjunction	22	84.6%	21.2%
Determiner	24	84%	14.4%
Article	24	84.6%	14.3%
Preposition	25	96.2%	10.9%
Verb	28	98.7%	0.8%
Proper Noun	28	100%	0%
Overall		95.7%	5.1%

The headline is not the 5.1% overall saving but the structural pattern it reveals: the model allocates the same compute to every token regardless of difficulty, and LayerLens makes that mismatch measurable for the first time without modifying the model.

Cross-Model Comparison

To test whether POS-layer specialization is specific to Qwen3-0.6B or a general transformer property, the same analysis was run on GPT-2 and the two fingerprint matrices were compared using cosine similarity and Dynamic Time Warping. Figure 6 shows the fingerprint heatmaps for both models side by side.

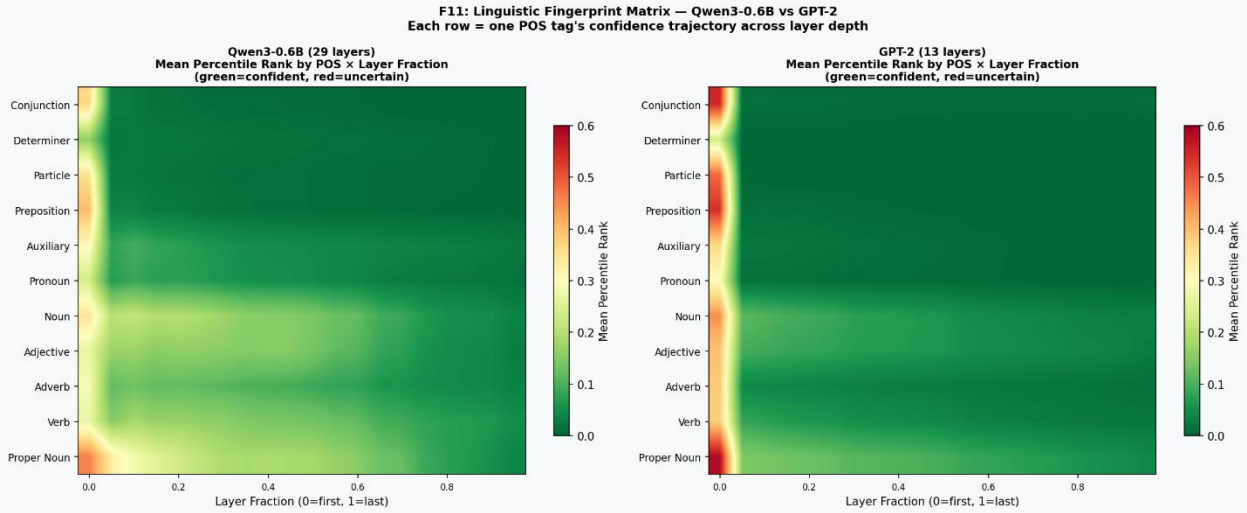

Figure 6 Linguistic fingerprint heatmaps for Qwen3-0.6B and GPT-2

The mean cosine similarity across all 11 POS categories is 0.897 and the mean DTW distance is 0.201. All 11 categories are consistent with the interpretation that both models run the same linguistic pipeline at different speeds. Function words show the tightest agreement (conjunctions: cosine 0.997, DTW 0.183). Content words show slightly more variation (verbs: cosine 0.799, DTW 0.276), but the same directional pattern holds. POS-layer specialization is a property of transformer architecture generally, not an artifact of a single model or vocabulary size.

Conclusions

What was Solved

This project addressed three problems in transformer interpretability and uncertainty estimation, and produced concrete, quantified results on all three.

The first problem was the absence of a lightweight method to profile how a transformer model distributes linguistic processing across its layers. LayerLens solves this reproducibly using the logit lens with no additional training or labeled data. The result is a statistically confirmed finding: parts of speech saturate at systematically different layers across a 2,500-sentence corpus, with a gap of 5.9 layers between conjunctions and proper nouns. This ordering is consistent across news, scientific, creative, and social domains and is robust to sentence length, with Spearman rank correlations above 0.90 in every sentence length bucket. Cross-model Dynamic Time Warping analysis further confirms that this specialization is not a property of a single architecture: Qwen3-0.6B and GPT-2 show a mean cosine fingerprint similarity of 0.897 across 11 parts of speech, supporting the interpretation that POS-layer specialization is a general transformer property.

The second problem focused on improving confidence tracking, as final-layer softmax probability alone is often an unreliable indicator of whether a model's prediction is actually correct. LayerLens addresses this by replacing the final-layer view with a trajectory-based uncertainty score built from three features: monotonicity, entry speed, and persistence. On the full corpus of 32,453 natural language tokens, the learned trajectory score achieves an AUROC of 0.8151 compared to 0.7783 for softmax, a gain

of 0.037. The Late Rise trajectory type highlights a major blind spot in standard inference: because final-layer softmax has no memory of the computation history, it mistakes sudden, last-second rank jumps for genuine convergence. However, LayerLens shows these tokens are highly unstable, resulting in an 86.1% error rate under soft correctness scoring. Detecting this specific pattern gives developers a concrete, trajectory-based signal to intercept unreliable predictions at marginal computational cost.

The third problem was the lack of visibility into compute allocation efficiency. The early exit simulation shows that 21 percent of compute spent on conjunctions produces no change in the model's final prediction. This quantifies a structural mismatch between word complexity and compute allocation that is built into standard transformer inference and makes it measurable for the first time without any modification to the model.

What Remains Open

Several questions were only partially answered or remain unresolved. The early exit simulation is observational: it measures whether a token's own prediction changes when its hidden state is frozen at the saturation layer, but it does not account for the effect of that frozen state on subsequent tokens through the attention mechanism. A true causal test would require re-running the full autoregressive KV-cache with frozen states, which was outside the scope of this project.

The soft correctness Stage 2, which uses cosine similarity in the model's input embedding space, produced an inverted calibration result: antonym pairs were more similar than synonym pairs in Qwen3-0.6B's static embeddings. This is a known property of distributional embeddings rather than a bug in the system, but it means the semantic similarity stage introduces noise and its results cannot be cited with confidence for individual POS claims, particularly for verbs.

Analysis of the 0.4–0.6 softmax probability band defines the optimal deployment scope for the framework. While LayerLens delivers a consistent overall AUROC improvement of 0.037 across the full corpus, standard softmax maintains better separation specifically when the model is in a state of moderate mid-range ambiguity (0.8373 vs. 0.6006). This behavior indicates that the trajectory score functions best as a high-utility complementary feature in a hybrid confidence pipeline, rather than a standalone replacement for softmax.

Suggestions for Future Work

The most direct extension of this work is replacing the static input embeddings used in soft correctness Stage 2 with contextual hidden-state embeddings. Using the model's own intermediate representations rather than its input embedding matrix would likely produce more semantically grounded similarity thresholds and eliminate the antonym inversion problem observed in this project.

A full causal early exit experiment, in which the KV-cache is re-run with hidden states frozen at the saturation layer, would convert the current observational result into a true computational claim. This would allow LayerLens to report not just that a token's own prediction does not change after saturation, but that subsequent tokens are also unaffected, making the compute savings argument significantly stronger.

Extending the analysis to larger models such as Llama, Mistral, or GPT-4 class architectures would test the universality claim at scale. The current DTW result, showing consistent POS processing between a 13-layer and a 28-layer model, is a promising foundation, but the claim becomes much stronger if it holds across models with 32, 64, or 96 layers. Applying LayerLens to encoder-only models would additionally test whether the saturation and trajectory patterns observed here are specific to the autoregressive language modeling objective or are a more general property of the attention mechanism.

A significant architecture optimization for the inference pipeline would be replacing the vanilla LogitLens projection with a TunedLens framework (Belrose et al., 2023) . While the original LogitLens was chosen for its zero-training utility , its early-layer representations suffer from scaling noise that required this project to restrict aggregate statistics to layers 10 and above . Integrating a TunedLens, which applies a lightweight, learned affine correction at each layer, would stabilize these early intermediate hidden states. This upgrade would expand the system’s capabilities by

allowing LayerLens to track clean, reliable token trajectories from layer 1 onward, unlocking visibility into the model's earliest processing steps.

Finally, the trajectory-based uncertainty score could be integrated directly into model serving infrastructure as a lightweight post-processing step. Because it operates on hidden states already computed during inference and requires no additional forward passes, it could provide per-token confidence estimates at marginal cost. This would be particularly valuable in retrieval-augmented generation and structured output settings where token-level reliability matters.

References

Barbieri, F., Camacho-Collados, J., Espinosa Anke, L., & Neves, L. (2020). TweetEval: Unified benchmark and comparative evaluation for tweet classification. Findings of the Association for Computational Linguistics: EMNLP 2020, 1644–1650.

Belrose, N., Furman, Z., Smith, L., Halawi, D., Oikarinen, T., & Song, L. (2023). Eliciting latent predictions from transformers with the tuned lens. arXiv preprint arXiv:2303.08112.

Belinkov, Y. (2022). Probing classifiers: Promises, shortcomings, and advances. Computational Linguistics, 48(1), 207–219.

Bird, S., Klein, E., & Loper, E. (2009). Natural language processing with Python. O'Reilly Media.

Eldan, R., & Li, Y. (2023). TinyStories: How small can language models be and still speak coherent English? arXiv preprint arXiv:2305.07759.

Guo, C., Pleiss, G., Sun, Y., & Weinberger, K. Q. (2017). On calibration of modern neural networks. Proceedings of the 34th International Conference on Machine Learning, 1321–1330.

Honnibal, M., Montani, I., Van Landeghem, S., & Boyd, A. (2020). spaCy: Industrial-strength natural language processing in Python. Zenodo. https://doi.org/10.5281/zenodo.1212303

Husain, H., Wu, H. H., Gazit, T., Allamanis, M., & Brockschmidt, M. (2019). CodeSearchNet challenge: Evaluating the state of semantic code search. arXiv preprint arXiv:1909.09436.

Jin, Q., Dhingra, B., Liu, Z., Cohen, W., & Lu, X. (2019). PubMedQA: A dataset for biomedical research question answering. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, 2567–2577.

Meert, W., Hendrickx, K., & Van Craenendonck, T. (2020). dtaidistance (Version 2) [Software]. GitHub. https://github.com/wannesm/dtaidistance

Miller, G. A. (1995). WordNet: A lexical database for English. Communications of the ACM, 38(11), 39–41.

Nostalgebraist(identity unknown). (2020). Interpreting GPT: The logit lens. LessWrong. https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesneau, E. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825–2830.

Tenney, I., Das, D., & Pavlick, E. (2019). BERT rediscovers the classical NLP pipeline. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 4593–4601.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30, 5998–6008.

Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. Advances in Neural Information Processing Systems, 28, 649–657.