PROTOTYPE TRANSFORMER: TOWARDS LANGUAGE MODEL ARCHITECTURES INTERPRETABLE BY DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

While state-of-the-art language models (LMs) surpass the vast majority of humans in certain domains, their reasoning remains largely opaque, undermining trust in their output. Furthermore, while autoregressive LMs can output explicit reasoning, their true reasoning process is opaque, which introduces risks like deception and hallucination. In this work, we introduce the Prototype Transformer (ProtoT)—an autoregressive LM architecture based on prototypes (parameter vectors), posed as an alternative to the standard self-attention-based transformers. ProtoT works by means of two-way communication between the input sequence and the prototypes, and we show that this leads to the prototypes automatically capturing nameable concepts (e.g. "woman") during training. They provide the potential to interpret the model's reasoning and execute targeted edits of its behavior. Furthermore, by design, the prototypes create communication channels that aggregate contextual information at different time scales, aiding interpretability. In terms of computation scalability, ProtoT scales linearly with sequence length vs the quadratic scalability of SOTA self-attention transformers. Compared to baselines, ProtoT scales well with model and data size, and achieves good performance on downstream benchmarks (GLUE). ProtoT exhibits robustness to input perturbations on par or better than some baselines, but differs from them by providing interpretable pathways showing how robustness and sensitivity arises. Reaching close to the performance of state-of-the-art architectures, ProtoT paves the way towards creating well-performing autoregressive LMs interpretable by design.

1 INTRODUCTION

Large-scale autoregressive language models have achieved strong performance across various domains, with architectures like GPT-4 and LLaMA (Achiam et al., 2023; Touvron et al., 2023a) demonstrating capabilities on benchmarks spanning mathematical reasoning, code generation, and natural language understanding tasks. However, these models exhibit limited transparency in their reasoning processes, creating challenges for understanding how they arrive at their outputs and potentially limiting their deployment in applications where interpretability is important. For example, it has been observed that there is a large disconnect between models' explicit reasoning and their internal computational processes (Greenblatt et al., 2024): while language models can generate step-by-step explanations when prompted, research indicates that these explanations may not reflect their actual reasoning pathways (Turpin et al., 2023). This opacity also contributes to hallucination behaviors, where models generate confident but factually incorrect outputs without clear indicators of uncertainty (Zhang et al., 2025).

Current interpretability methods for language models primarily operate as post-hoc analysis tools on architectures not designed with interpretability as a primary consideration. Approaches such as attention visualization (Clark et al., 2019), probing techniques (Tenney et al., 2019), and causal intervention methods (Meng et al., 2022) provide insights into model behavior but face limitations imposed by the underlying self-attention architecture. More recent techniques like sparse autoencoders (Kissane et al., 2024) attempt to disentangle superposed features within existing architectures, though they still operate within the constraints of standard transformer designs.

In this work, we present the Prototype Transformer (ProtoT), an alternative autoregressive language model architecture that incorporates interpretability considerations directly into its design. ProtoT

replaces the standard self-attention mechanism with a prototype-based approach, where learnable parameter vectors engage in bidirectional communication with input sequences. This design choice allows prototypes to capture interpretable concepts during training, providing more direct access to the model's reasoning components.

ProtoT offers several characteristics that distinguish it from standard transformer architectures. The prototype-based design enables direct inspection and modification of learned concepts, supporting targeted behavioral adjustments based on identifiable reasoning components. The architecture aggregates contextual information across different temporal scales through prototype communication channels, which facilitates the interpretation of both local and global reasoning patterns. Additionally, ProtoT operates with linear computational complexity relative to sequence length, versus quadratic for the standard self-attention. The explicit prototype representations also enable more direct causal attribution by connecting specific predictions to their contributing components. Our contributions are briefly as follows:

- We introduce ProtoT, a novel autoregressive language model architecture that replaces self-attention with prototype-based communication. The architecture uses learnable parameter vectors that engage in bidirectional communication with input sequences, achieving linear computational complexity while maintaining competitive performance.
- We demonstrate that prototypes automatically learn coherent, nameable concepts during training, enabling direct interpretation of model reasoning. Our analysis reveals that prototypes capture disentangled semantic concepts across abstraction levels, with intervention experiments showing measurable causal effects on "predict and consolidate" behavior patterns.
- We provide extensive evaluation showing that ProtoT achieves competitive GLUE performance while offering superior robustness analysis. The architecture demonstrates appropriate stability under meaning-preserving perturbations and semantic sensitivity under interventions, with prototype-mediated robustness providing interpretable pathways for understanding behavior.

2 Related Work

One of the main goals when it comes to interpreting language models is to identify which components (e.g., heads, layers, neurons) are responsible for specific outputs or behaviors of the model (Zhang & Nanda, 2023). This is a complex task, as it is not possible to simply look at the magnitude of the attention head: does not necessarily imply causal importance, as it has been shown that such weights can often be perturbed without changing downstream behavior (Jain & Wallace, 2019). Moreover, due to *superposition*, where multiple concepts or features are encoded together in overlapping neurons or activation directions—it becomes difficult to isolate representations for individual concepts (Elhage et al., 2022). To this end, this problem is often tackled via causal intervention, where we analyze the difference in components between a clean and a corrupted prompt, as well as how the output changes when such activations are swapped (Meng et al., 2022). This technique has been used to understand where information is stored both in individual weights and layers (Geva et al., 2023), and in larger subnetworks (Wang et al., 2022). One promising approach in this domain is sparse autoencoders (SAEs) applied not just to MLPs or residual streams but, more recently, to attention layer outputs (Kissane et al., 2024). The goal of this line of work is to disentangle individual features embedded within activations and attention heads, so that superposed features become more separable and interpretable (Rai et al., 2024).

Prototype methods in NLP seek to render model decisions interpretable by relating inputs to prototypical examples, following the idea from computer vision, where the model learns prototypical parts of images and compares them to input images for classification (Chen et al., 2019). Recently in computer vision, ProtoViT (Ma et al., 2024) integrates Vision Transformer backbones with adaptive prototype-based learning, using prototypes as deformable parts in a final classification layer for case-based reasoning, demonstrating how prototype methods can be successfully adapted to modern transformer architectures. More generally, works in computer vision all have prototype features in the final layer (Rymarczyk et al., 2022), whereas our architecture learns prototype features at every level of the hierarchy. In NLP, ProtoAttend (Arik & Pfister, 2020) uses attention over learned prototypes (entire training examples) so that decisions are made via similarity, offering sample-based interpretability with modest accuracy loss. ProtoryNet (Hong et al., 2023) introduces prototype trajectories, mapping each sentence in a multi-sentence input to a prototype and modeling how these

activations evolve, for finer-grained interpretability and reduced prototype set size. ProSeNet adapts prototype reasoning for text classification with constraints for sparsity/diversity so that prototypes are more meaningful, and allows inspection of which prototypes are most relevant (Meng et al., 2022). To conclude, two very recent works are ProtoLens, that advance prototype interpretability by going to sub-sentence span extraction (Wei & Zhu, 2025), and a white box framework for sarcasm detection, where prototype tensors are built over transformer embeddings and sentiment embeddings (Wen & Rezapour, 2025).

Recent work has explored alternatives to self-attention using fixed sets of latent vectors for information routing. Slot Attention (Locatello et al., 2020) uses iterative competitive binding where slots compete to capture distinct entities, while the Perceiver family employs latents as processing bottlenecks—with Perceiver IO (Jaegle et al., 2022) using bidirectional cross-attention and Perceiver AR (Hawthorne et al., 2022) adding causal masking for autoregressive tasks. Our prototype-based approach differs fundamentally by treating proto-s as semantic routing vectors that create R distinct communication channels, each with separate read/write gates and learnable time-discount param-s.

3 PROTOTYPE TRANSFORMER

The prototype transformer (ProtoT) is an autoregressive LM architecture, based on prototypes. It is a transformer stack similar to LLaMA-3 (Grattafiori et al., 2024), with a mixer and feed-forward module, skip-connections, and RMSNorm, with the only difference in the mixer module. Instead of standard self-attention, ProtoT uses a prototype-based mixer.

Prototype mixer: This module is a self-attention alternative that uses R prototypes to route the communication across the sequence through R corresponding channels, according to the dot-products with the prototypes (read gate) (Eq. 1). Each communication channel is used to aggregate information from the past, modulated by the dot-products between the prototype for that channel and the tokens (write gate). A per-channel learned time-discount β_k is applied to modulate the time sensitivity of each channel. On a high level, the prototype mixer follows the following formula (simplified for clarity) applied at any token position i, for linear maps U, V, and W, with full details below:

$$x_i \leftarrow U\Big(\sum_{k=1,\dots,R} \underbrace{(W(x_i)\cdot \mathbf{P}_k)}_{\text{Read Gate}} \underbrace{\sum_{j<i} \underbrace{\beta_k^{i-j}}_{\text{Time Discount}} \underbrace{(x_j \cdot \mathbf{P}_k)}_{\text{Write Gate}} V(x_j)}_{\text{Prefix Mean}} \Big) \tag{1}$$

This mixer can be interpreted as splitting a token representation into R semantic channels (via prototypes), then for each channel, aggregating semantic information from past tokens by using the corresponding prototype as a filter. Note the strict causality (past-only aggregation, over x_j for $j < i$ only) in Eq. 1, in contrast to self-attention where each token position can attend to itself. See *Prefix Mean* below for more details. We also use *low-rank projection* at $1/2$ of the hidden size (h) at the value path ($V(x_j)$), which saves up to 50% compute at the mixer module, at a very minor perplexity cost. We keep the prototypes and routing (read and write gates) in the full size h as they have only linear compute in h. Each mixer module is followed by a single feed-forward module, transformer-style, in L blocks ("layers"). Following LLaMA-3 (Grattafiori et al., 2024), we use SwiGLU (Shazeer, 2020) with the same intermediate ratio of $\sim$2.7. We also use RMS pre-layernorm (Zhang & Sennrich, 2019) around each of the Proto Mixer and MLP, as in LLaMA.

Prototypes. They are R parameter vectors of dimension the model's hidden size h. They are shared between the *read gate* and *write gate* in Eq. 1 via dot-products with the contextualized sequence (x_i). As prototypes are a fixed number of vectors directly communicating with the contextualized token embeddings, they can capture concepts as vectors, which we demonstrate in Section 5.1.

Write gate: cross-attention between prototypes P_k and contextualized token representations x_j: $\prod_W = softmax_k((x_j \cdot \mathbf{P}_k)/\tau_w)$, with softmax over the prototypes and a learned temperature τ_w. It works as follows: each sequence member (x_j), after communicating with the prototypes in $\prod_W$, writes to the communication channels corresponding to those prototypes (with P_k serving as filters). A *local convolution* at the values stream (immediately after $V(x_j)$) at layers 0 and 1 applies a convolution across the 4 past tokens and the current one, with one channel per each unit of hidden

dimension h. It adds expressivity at the value stream, by capturing short-term relationships, and we observed reduced perplexity and increased utility of layer 0 (as measured by increased alpha-gate).

Read gate: a cross-attention-like mechanism that routes communication to the R channels according to similarity with the corresponding R prototypes. It is identical to the write gate (above), except for the linear map $W(x_i)$ and separate temperature τ_r: $\prod_W = softmax_k((W(x_j) \cdot \mathbf{P}_k)/\tau_r)$, which add expressivity and reduce perplexity. Using this gate (vs. using a copy of the write gate) may also help with interpretability, by decoupling the interpretability of channel routing from the interpretability of sequence aggregation. At layer 0 we use *shared gating* (i.e., removing the W linear map in Eq. 1) which we observed reduces perplexity and increases utility (higher alpha-gate value, see definition below), likely by providing an inductive bias that reduces noise.

Prefix mean: a cumulative sum operation (as shown in Eq. 1) which enforces causality in the LM (for next-token-prediction): at any given sequence position, entries are summed for all previous positions ($j < i$), so the model cannot cheat in the next-token-prediction task. Furthermore, this is a stronger causality constraint than self-attention, where each positions can also attend to itself. Removing this direct vertical path aims to encourage the write gate to anticipate (predict) the read gate, and as we show in Section 5.1, this is exactly what happens. The prefix mean has *R communication channels*, each corresponding to a prototype. This allows it to aggregate long-context features in a more expressive ways (versus using one channel). A *discounted prefix* gives per-channel time preference, allowing aggregation at different time scales across channels. It is defined an exponential moving average (EMA) (time discount) on the Prefix Mean (Eq. 1), $\beta_k = \sigma(\gamma_k) \in (0, 1)$, parametrized by learnable γ. It can also be used to interpret the time preference of each prototype, as in Section 5.1. *Mass normalization* then turns the prefix cumulative sum into a prefix mean by dividing it by the sum of coefficients $\sum_{j<i} \beta_k^{i-j}(x_j \cdot \mathbf{P}_k)$. This theoretically stabilizes the computation, and we have observed reduced perplexity at only minor computational cost.

Alpha Gate: a ReZero-like (Bachlechner et al., 2020) scalar gate applied at the output of each Prototype Mixer module before it merges with the residual stream (skip-connection). It modulates the contribution of the mixer to the residual stream, slightly reducing perplexity at almost no computational cost. It also helps with debugging the model: a low value of alpha at a given layer (declining rapidly over the course of training) is strong evidence that the mixer is not contributing to the overall prediction. We initialize $\alpha = 1.0$ (versus ReZero's 0.0), which we found works best for ProtoT.

Compute: The ProtoT computational cost scales linearly in sequence length, as visible by Eq. 1. Note the recurrence in Eq. 1: the Prefix Mean for x_i depends only on the Prefix Mean for x_{i-1} and on x_{i-1}, both of which can be cached. This means that the model can generate tokens at sequence-wise constant (O(1)) computational and memory cost.

Ablation note. We ablate the layer-0 stabilizers—shared read/write routing, the layer-0 temperature init τ_0=3.0, and a k=5 write-value convolution—in Table 10 (Appendix). The all-mitigations setting is our ProtoT baseline: removing only the convolution degrades perplexity by $+12.4$ and halves $\bar{\alpha}_0$, while disabling all three reproduces the layer-0 collapse.

4 Experimental Setup

Baselines: We compare Prototype Transformer (ProtoT) to three representative mixer families while keeping the backbone fixed: depth (6), hidden size (256), FFN ratio (2.7×), RMSNorm, dropout (0.1), and the training recipe. We use the same tokenizer and optimizer across models and do not reuse any pre-trained weights. To isolate mixer effects, we exclude MoE and hybrid architectures (e.g., Jamba) (Lieber et al., 2024). We compare against a *LLaMA-style Transformer*: a single-expert, decoder-only self-attention Transformer in the LLaMA-3/3.1 style (Grattafiori et al., 2024), matched to ProtoT in backbone hyperparameters and training setup. The only architectural difference is the mixer (self-attention vs. prototype); *Mamba*: a modern state-space model (SSM) instantiation with the same dimensionality (6 layers, hidden 256) and FFN configuration as ProtoT (Gu & Dao, 2023) and *DeltaNet (delta-rule linear transformer)*: a linear-attention baseline following the DeltaNet formulation, configured with the same width, depth, and FFN ratio as ProtoT (Yang et al., 2024).

Throughput Benchmarks: We evaluate training throughput under identical conditions (same data pipeline, optimizer, precision BF16, sequence length 256, batch sizes 32 and 128). ProtoT achieves 25.2 and 7.6 it/s (batch 32/128), outperforming Mamba (11.9 and 3.2 it/s) and DeltaNet (3.5 and 1.8 it/s) while lagging behind the optimized LLaMA attention baseline (55.1 and 23.6 it/s). See Appendix A.6 for detailed benchmarks and FLOP analyses.

Dataset: We use a subset of the FineWeb-Edu dataset (Penedo et al., 2024), a high-quality web crawl dataset specifically curated for language model training. FineWeb-Edu consists of educational and informational web pages, providing diverse, coherent text that is well-suited for training autoregressive language models. The full dataset contains approximately 1.3 trillion tokens (with a less strict "score-2" filtering version containing 5.4 trillion tokens), but we use a manageable 250 million token subset for our experiments. Our sampled dataset contains 360,313 documents, with an average document length of 694 tokens. We use a custom BPE tokenizer (Sennrich et al., 2015) trained on the dataset with a vocabulary size of 16,000 tokens. For training, we use 338,695 documents (234.9M tokens) for the train split, 18,015 documents (12.5M tokens) for validation, and 3,603 documents (2.6M tokens) for testing. We chose this dataset over alternatives like C4 or The Pile because of its focus on high-quality, educational content.

Hyperparameter search: We do hyperparameter search on 18k examples for 10 epochs of the training data, with the default model sizes: hidden size h=256, layers L=6, and context ctx=256, unless otherwise specified. We use automatic search over batch size (32, 64, 128) and learning rate (from interval (3e-5, 3e-2)). For the search, we use Optuna with BoTorchSampler, with 15-trial warmup and 50 total trials, averaging over 3 seeds per trial. We found that batch size of 32 works best for all, but that different models have different best LR. See Appendix A.2 for more details.

Learning rate and scheduler: We use linear warmup over 2% of training, and cosine annealing towards 10% of the peak learning rate. This is common practice in large LM training (Kalra & Barkeshli, 2024) for two reasons: (1) warmup helps reduce divergence for large LMs (e.g., we observed that LLaMA had convergence issues without warmup when we tried the large-scale setting), and (2) cosine annealing helps with reaching higher peaks and lower lows of the LR (e.g., we found that values more than 1.6e-3 were best, compared to 7e-4 for flat LR), and lead to significantly lower perplexity numbers (2-6 perplexity points lower, depending on the model, in the default data and model size setting). We train all our models with AdamW (Loshchilov & Hutter, 2017), following standard practice in language modeling. Compared to SGD, AdamW is more robust to hyp. choice (Zhao et al., 2024).

Dropout: For all models, we use dropout (with probability 0.1) after the token embeddings, at the residual (block output) between blocks, and inside the FFN, because we find that it reduces ppl for all models. This is likely because it prevents overfitting in the multi-epoch training regime (10 epochs) that we use. For LLaMA, we additionally put dropout inside the self-attention (HuggingFace-supported option), which further decreases ppl slightly.

Attention heads and prototypes (R): Similar to (Press & Wolf, 2017), we have found that sharing the weights between embeddings and LM head reduces perplexity at the hyperparameter search stage, for all models. This is likely because it provides a good inductive bias aligning the token embeddings between input and final projection. We keep this choice at large-scale experiments as well, for simplicity. We also select attention heads from $\{2, 4, 8\}$, but at both small-scale and large-scale runs we have found that 4 works best for all models with attention heads (LLaMA and DeltaNet), which is what we use. For ProtoT's prototypes (R), we have found diminishing returns in terms of perplexity improvements beyond R=32, while computation scales linearly with R. Therefore, we use R=32 for all runs. Unless noted, all ProtoT results use the stabilized layer-0 router; see Table 10.

5 EXPERIMENTS

Large-scale training: In Table 1, we compare ProtoT to the 3 baselines at large-scale training (first vs. last column). We study the effect of simultaneously scaling the hidden size 2x, the layers 2x, the context size by 2x, and the training data $\sim$19x, versus the default training settings. The results

Table 1: Long-context scalability: *Cols. 1–4:* scaling from the default 256 up to 2048; *Cols. 1 & 5:* Default (h=256, L=6, ctx=256, Ex=18k) vs. Large-scale training (h=512, L=12, ctx=512, Ex=339k). Perplexity (lower is better). Best result in each section in bold.

Model	Default	512	1024	2048	Large-scale
LLaMA	**81.2**	**72.6**	**67.4**	**65.6**	**26.2**
Mamba	88.9	80.6	73.0	71.8	26.9
DeltaNet	93.5	78.6	72.9	71.2	32.1
ProtoT	93.6	87.5	83.4	84.6	30.0
ProtoT (h=512)	100.5	84.8	**76.3**	**75.4**	–
ProtoT (L=12)	113.3	**82.5**	77.2	79.3	–
ProtoT (R=64)	97.3	86.0	81.8	83.1	–

show that ProtoT scales well to the large model/data scenario. We show that ProtoT maintains relative performance to LLaMA, or even improves it (15.3 $\rightarrow$ 14.5% worse) with scale. Furthermore, ProtoT outperforms the DeltaNet linear-attention baseline (30.0 vs. 32.1 ppl, respectively). However, a large gap remains versus LLaMA and the Mamba state-space model (30.0 vs. 26.2 and vs. 26.9, respectively). While we did our best to optimise ProtoT, this is the first iteration of the model, whereas established LMs like LLaMA have had multiple (Touvron et al., 2023a;b; Grattafiori et al., 2024). We expect with community feedback and further refinement to shrink this gap.

Long-context scalability: The results in Table 1 (columns 1-4) show that ProtoT scales poorly with context length (if other model dimensions are fixed), which suggests that ProtoT is running into a bottleneck. This is likely because the cross-sequence communications pass through the prefix mean (Eq. 1), over R channels with h hidden dimensions each, which can be restrictive. We further investigate this issue in the final 3 rows of Table 1, where we compare possible culprits: the hidden size h, the number of prototypes R, and the layers L (which can also play a role). The results show that the hidden dimension is the most restrictive as increasing it is the only one of the 3 that keeps improving with context size beyond 1024. Our model is most affected by this likely because of our choice to project down to $h/2$ at the values ($V(x_j)$ in Eq.1) to save compute, further exacerbating this bottleneck. In practice, this is less of an issue because, in more realistic settings (e.g. *Large-Scale Training*), the larger capacity of the model would allow for larger context lengths.

Downstream performance: To comprehensively evaluate the general-purpose language understanding of ProtoT vs baselines, we fine-tune on the GLUE benchmark (Wang et al., 2018) consisting of 9 English NLU tasks spanning sentence- and sentence-pair classification as well as semantic textual similarity (more details in Appendix A.4).

As shown in Table 2, Overall, LLaMA performs best among the compared models, while ProtoT attains first- or second-place results on multiple tasks, demonstrating stable overall performance. Specifically, ProtoT achieves the best score on RTE, indicating a structural advantage in natural language inference under small-sample/low-resource settings; it obtains the second-best score on COLA, reflecting strong sensitivity to syntactic/local structural cues; and on mainstream single-sentence/sentence-pair classification tasks such as SST-2 and QNLI, ProtoT maintains consistently strong performance. For QQP/MRPC (sentence-pair paraphrase/duplicate detection), ProtoT is slightly behind the top system yet remains competitive. Moreover, ProtoT demonstrates strong cross-domain robustness on MNLI/MNLI-mm. Taken together with its best result on RTE (a low-resource NLI task), we believe ProtoT can maintain stable competitiveness on large-scale inference benchmarks while exhibiting structural advantages in low-resource settings.

Table 2: GLUE dev downstream fine-tuning results (all metrics reported as percentages). For COLA we report Matthews correlation; for SST-2 accuracy; for MRPC F1; for STS-B Pearson correlation; for RTE, WNLI, QNLI, MNLI and MNLI-MM accuracy; for QQP F1. Best results are in bold.

Model	COLA	SST-2	MRPC	STS-B	RTE	WNLI	QQP	QNLI	MNLI	MNLI-MM
LLaMA	**36.2**	**92.3**	**85.3**	**85.7**	55.6	46.5	**85.7**	**86.3**	**79.6**	**80.6**
Mamba	31.7	89.2	82.5	78.3	55.6	**56.3**	82.4	82.3	75.2	75.8
DeltaNet	14.2	85.3	81.5	75.7	53.4	43.7	81.3	80.6	71.7	72.7
ProtoT	32.7	90.0	81.9	75.1	**58.1**	**56.3**	81.6	82.4	75.4	74.9

5.1 INTERPRETABILITY

Prototypes act as explicit representational slots, with each token routed into R prototype channels through the read gate and updating them via the write gate (Eq. 1). This discrete structure allows features to be stored and reused across the sequence, enabling association of prototypes with identifiable concepts. Each prototype also has an associated decay parameter β_k, applied in the prefix mean (Eq. 1) to discount past activations. Larger β_k values produce faster decay, while smaller values allow information to persist longer. For interpretability, we report the derived half-life $t_{1/2}^{(k)} = \ln(2)/\beta_k$, specifying the expected number of steps for a prototype's contribution to halve, providing a direct way to analyze specialization in short- or long-term dependencies. By analyzing the interactions between read and write gates, we inspect *Generation behavior*—how the model integrates context for next-token prediction. Studying the resulting activation patterns reveals the read-write dynamics that guide token generation and shows how the model leverages prototype representations to process sequences effectively.

Experiments: To investigate intepretability properties, we design three experiments. For concepts and half-life, we compute write routing activations across sequences from the FineWeb validation set for each prototype, aggregate them at the sequence level, and rank sequences by total activation strength. This identifies sequences that most strongly activate each prototype and allows to visually inspect learned concepts and relation between temporal locality and β_k parameter.

We also analyze the alternation of write and read phases across the sequence. For a subset of prototypes, we select the most activating sequences and compute write and read routing activations for each token along the same prototype. This enables inspection of the internal dynamics of ProtoT, showing how sequence level information is aggregated and maintained during processing. We present illustrative examples in Figure 1. Additional examples are included in Figures 5, 6, 7, 8, and 9.

We probe the functional role of individual prototypes through a targeted intervention experiment. Based on 'write' gate activations from the FineWeb validation set, we identified three functionally distinct prototypes from Layer 9: *L9 P7*, which encodes a 'female' concept; *L9 P18*, which partially encodes a 'male' concept; and *L9 P2*, a gender-neutral control. Our intervention consists of disrupting each of these prototypes via parameter re-initialization and measuring the subsequent change in the conditional probability of the target words 'women' and 'girls'. We illustrate those prototypes in Figures 2, 3, and 4. Additional details on the construction of test sentences are in Appendix A.3.1.

Write gate activations results: Analysis reveals that prototypes capture disentangled concepts across varying levels of semantic abstraction, which naturally emerge as a result of training and encode interpretable patterns. For example, we can identify concepts like entity names, functional words, verbs, as well as composite dates, illnesses, or school-related narratives. We also find that these concepts generally reflect the hierarchical organization of the model, with early layers tending to capture more superficial patterns and deeper layers representing composite and abstract semantics. Polysemanticity is present in a few prototypes but remains limited overall. Furthermore, we identify a correlation between half-life values and the encoded concepts, where lower half-life values tend to capture local elements (like entities, stop words, or punctuation). These results show that dual-channel communication forms prototype hubs that can largely be treated as separate, disentangled concept hubs, highlighting their potential for interpretability.

Write-read alternation pattern results: We observe a consistent temporal pattern in read and write activations, with read activity peaking one step before write activity. For example, in the results shown in Figure 1 (right), for the token 'protection', the read gate activates prototype 4 at the preceding token 'fall', followed by write activation on 'protection'. This pattern is consistently seen across the most strongly activating sequences for each prototype and suggests that read and write gates may develop coordinated interactions. Such coordination is consistent with a "predict and consolidate" behavior, where the read gate appears to anticipate which prototype may be relevant for upcoming tokens, and the write gate subsequently updates the memory based on the current token.

Prototype intervention results: Our intervention experiments, demonstrate that prototypes function as specific and interacting semantic hubs. Disrupting the 'female' prototype *L9 P7* significantly

decreased the probability of related words (e.g., -17.80% for 'women' in seed sentence), highlighting its functional importance for this concept. The specificity of this effect was validated by the negligible impact of disrupting the control prototype, *L9 P2*, as disrupting the 'male' prototype *L9 P18* consistently increased the probability of female-coded words (e.g., $+11.50\%$ for 'women' in seed sentence). These findings indicate that the model learns functionally distinct prototypes and uses them in an interactive manner to refine its predictions. Additional examples and detailed descriptions are in Appendix A.3.1.

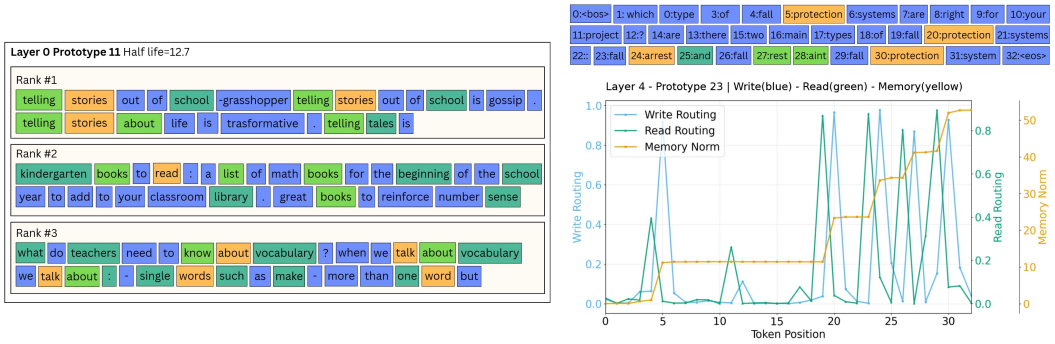

Figure 1: Left: Sequences most strongly activating prototype 11 at layer 0, which encodes the concept of narrative in a scholastic context. Right: Write-gate, read-gate, and memory curves for a sequence that strongly activates prototype 23 at layer 4. Read-gate peaks precede write-gate activations, spiking on the token immediately before those that trigger write-gate routing.

5.2 Robustness

We analyse robustness of ProtoT from three complementary perspectives: (1) robustness to *meaning-preserving noise perturbations*, (2) robustness to *prototype clamping perturbations*, and (3) robustness to *causal interventions* that alter semantics. This unified view clarifies both stability under benign variations and sensitivity to intended changes.

Noise perturbations: We first consider *black-box, surface-level perturbations* that preserve meaning (e.g., synonyms, typos, contractions). The perturbation benchmark (Appendix A.5) contains 3,500 semantically equivalent sentence pairs across seven categories. Robustness is quantified by the Jensen–Shannon divergence $JS(p(\cdot|x), p(\cdot|x'))$ between next-token distributions for an original input x and its perturbed variant x'. Lower values indicate greater stability. Table 3 shows that Mamba attains the lowest JS overall, confirming strong noise robustness. ProtoT, however, consistently outperforms LLaMA on synonyms, typos, and spelling, and closely matches it on morphology. This aligns with ProtoT's design: prototypes aggregate contextual information into nameable concepts, yielding stability under lexical variation. While ProtoT lags LLaMA on punctuation (where precise attention alignment is beneficial), it reliably surpasses DeltaNet and is overall competitive with strong baselines.

Table 3: Slice-level robustness measured by Jensen–Shannon divergence (lower is better). Abbreviations: *abbr.*=abbreviation, *contr.*=contraction, *morph.*=morphology, *punct.*=punctuation, *spell.*=spelling, *syn.*=synonym, *typo*=typos. Best per column in bold.

Model	abbr.	contr.	morph.	punct.	spell.	syn.	typo
DeltaNet	1.0657	0.8310	0.3784	0.5804	0.3547	0.6363	0.6257
LLaMA	0.3325	0.0449	0.0542	**0.1740**	0.0634	0.1450	0.2269
Mamba	**0.1441**	**0.0104**	**0.0308**	0.4428	**0.0054**	**0.0130**	**0.0761**
ProtoT	0.4166	0.0823	0.0542	0.3982	0.0260	0.1132	0.2074

Prototype clamping: To test whether robustness is mediated by prototype routing, we compute *Prototype-Mediated Robustness (PMR)*. For a pair (x, x'), let $JS_{\text{base}} = JS(p(\cdot|x), p(\cdot|x'))$. We then clamp the prototype routing weights from x onto x' and recompute $JS_{\text{clamped}} = JS(p(\cdot|x), p^{\text{clamped}}(\cdot|x'))$. We define $PMR = (JS_{\text{base}} - JS_{\text{clamped}})/JS_{\text{base}}$. A positive PMR indicates that prototypes mediate robustness, while negative values suggest residual pathways dominate.

Table 4 shows that mean PMR is sometimes slightly negative, but the positive fraction $PMR_{>0}$ is consistently 0.5–0.6, and $JS_{\text{clamped}} < JS_{\text{base}}$ across slices. This demonstrates that prototypes systematically contribute to robustness, providing interpretable routing pathways rather than opaque head-level aggregation.

Table 4: Prototype-Mediated Robustness (PMR). Mean and std of $\mathbf{PMR}$, fraction of positive cases, and average JSDs. Best per column in bold.

Slice	$\mathbf{PMR}_{\text{mean}}$	$\mathbf{PMR}_{\text{std}}$	$\mathbf{PMR}_{>0}$	$\mathbf{JS}_{\text{base}}$	$\mathbf{JS}_{\text{clamped}}$	**n**
abbreviation	-0.093	0.367	0.596	0.417	0.415	500
contraction	-0.027	0.104	0.330	0.082	0.083	500
morphology	-0.102	0.689	0.426	0.054	0.047	500
punctuation	-0.000	0.373	0.554	0.398	0.322	500
spelling	-0.033	0.225	**0.610**	**0.026**	**0.025**	500
synonym	**0.013**	**0.075**	0.606	0.113	0.109	500
typo	0.001	0.279	0.533	0.208	0.186	500

Intervention robustness: Finally, we study robustness under *causal interventions* that alter semantics: gender, negation, and number tags. Unlike surface perturbations, these flips should change predictions. We measure JS, top-k overlap (Ov), Spearman correlation (Sp), and top-1 invariance (T1). Higher JS and lower Ov/Sp/T1 indicate greater sensitivity to the intervention. Table 5 shows that while DeltaNet attains the highest raw JS, ProtoT consistently yields lower Ov, Sp, and T1 compared to LLaMA and Mamba. This indicates that ProtoT adapts more reliably under meaning-altering interventions, reflecting appropriate semantic sensitivity through prototype routing. LLaMA and Mamba often remain insensitive to such tags.

Table 5: Intervention robustness on gender (gen), negation (neg), and number (num). Metrics: JS (higher better), Ov/Sp/T1 (lower better). Best values in bold.

Model	**JS** (gen / neg / num)	**Ov** (gen / neg / num)	**Sp** (gen / neg / num)	**T1** (gen / neg / num)
DeltaNet	**0.054** / **0.173** / **0.282**	0.754 / **0.540** / **0.474**	0.610 / **0.176** / **0.033**	**0.616** / **0.388** / **0.330**
LLaMA	0.004 / 0.028 / 0.022	0.946 / 0.875 / 0.843	0.966 / 0.815 / 0.824	0.890 / 0.770 / 0.930
Mamba	0.003 / 0.006 / 0.007	0.936 / 0.935 / 0.907	0.949 / 0.910 / 0.907	0.884 / 0.992 / 0.948
ProtoT	0.037 / 0.081 / 0.083	**0.709** / 0.774 / 0.657	**0.429** / 0.536 / 0.441	0.690 / 0.806 / 0.806

Noise perturbation results establish that ProtoT is robust to lexical variation. PMR shows that prototypes actively mediate robustness, exposing interpretable mechanisms. Intervention robustness confirms that ProtoT distinguishes meaning-preserving from meaning-altering changes. Together, these findings show that ProtoT not only matches or surpasses baselines in robustness but also provides transparent pathways for analysing where robustness arises.

6 Conclusion

We have introduced the Prototype Transformer (ProtoT), an alternative autoregressive language model architecture that replaces standard self-attention mechanisms with prototype-based computation to enhance model interpretability. Through bidirectional communication between learnable prototype vectors and input sequences, ProtoT demonstrates that architectural design choices can support interpretability with only small compromise in performance. Our evaluation on GLUE benchmarks shows competitive results compared to standard transformers, while analysis reveals that prototypes automatically learn coherent, nameable concepts during training. The architecture also provides practical advantages through linear computational complexity and enables direct shape attribution of predictions to specific conceptual components and targeted editability.

Future work will study broader evaluation across diverse tasks and model scales, as well as a more fine-grained study of the interpretability benefits. In summary, our results show that incorporating interpretability considerations into architectural design may be compatible with maintaining competitive performance, though the full scope and boundaries of this approach require continued investigation. ProtoT contributes to ongoing research toward developing LMs that balance capability with transparency for applications where understanding and correcting model reasoning is essential.

7 Reproducibility Statement

We provide full details of the model architecture, training setup, and evaluation protocols in the main paper and appendix. The perturbation benchmark dataset (`perturbation[...].jsonl`), along with its generation and filtering scripts, is included in the supplementary material and will be released publicly upon acceptance. In addition, we introduce a manually constructed intervention benchmark dataset (`intervention_benchmark.jsonl`), which tests semantic interventions on gender, negation, and number. Since the dataset was curated directly rather than generated by scripts, we will release it in full to ensure exact reproducibility of the intervention robustness experiments. We also include the interactive html (`prototype_visualization_word_level.html`). All code to reproduce our experiments will likewise be made available upon acceptance.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Sercan O Arik and Tomas Pfister. Protoattend: Attention-based prototypical learning. *Journal of Machine Learning Research*, 21(210):1–35, 2020.

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Huanru Henry Mao, Garrison W. Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. *arXiv preprint arXiv:2003.04887*, 2020.

Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. This looks like that: deep learning for interpretable image recognition. *Advances in Neural Information Processing Systems*, 32, 2019.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. What does BERT look at? An analysis of BERT's attention. *arXiv preprint arXiv:1906.04341*, 2019.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

Mor Geva, Jasmijn Bastings, Katja Filippova, and Amir Globerson. Dissecting recall of factual associations in auto-regressive language models. *arXiv preprint arXiv:2304.14767*, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Ryan Greenblatt, Carson Denison, Benjamin Wright, Fabien Roger, Monte MacDiarmid, Sam Marks, Johannes Treutlein, Tim Belonax, Jack Chen, David Duvenaud, Akbir Khan, Julian Michael, Sören Mindermann, Ethan Perez, Linda Petrini, Jonathan Uesato, Jared Kaplan, Buck Shlegeris, Samuel R. Bowman, and Evan Hubinger. Alignment faking in large language models. *arXiv preprint arXiv:2412.14093*, 2024. URL `https://arxiv.org/abs/2412.14093`.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Curtis Hawthorne, Andrew Jaegle, Catalin Cangea, Sebastian Borgeaud, Charlie Nash, Mateusz Malinowski, Sander Dieleman, Oriol Vinyals, Matthew Botvinick, Ian Simon, et al. General-purpose, long-context autoregressive modeling with Perceiver AR. In *International Conference on Machine Learning*. PMLR, 2022.

Dat Hong, Tong Wang, and Stephen Baek. Protorynet-interpretable text classification via prototype trajectories. *Journal of Machine Learning Research*, 24(264):1–39, 2023.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier Hénaff, Matthew M Botvinick, Andrew Zisserman, Oriol Vinyals, and João Carreira. Perceiver IO: A general architecture for structured inputs & outputs. In *International Conference on Machine Learning*. PMLR, 2022.

Sarthak Jain and Byron C. Wallace. Attention is not explanation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019.

Dayal Singh Kalra and Maissam Barkeshli. Why warmup the learning rate? underlying mechanisms and improvements. *Advances in Neural Information Processing Systems*, 37:111760–111801, 2024.

Connor Kissane, Robert Krzyzanowski, Joseph Isaac Bloom, Arthur Conmy, and Neel Nanda. Interpreting attention layer outputs with sparse autoencoders. *arXiv preprint arXiv:2406.17759*, 2024.

Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

Shuaipeng Li, Penghao Zhao, Hailin Zhang, Xingwu Sun, Hao Wu, Dian Jiao, Weiyan Wang, Chengjun Liu, Zheng Fang, Jinbao Xue, Yangyu Tao, Bin Cui, and Di Wang. Surge phenomenon in optimal learning rate and batch size scaling. *Advances in Neural Information Processing Systems*, 37:132722–132746, 2024.

Yanran Li, Hui Su, Xiaoyu Shen, Wenjie Li, Ziqiang Cao, and Shuzi Niu. Dailydialog: A manually labelled multi-turn dialogue dataset. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (IJCNLP)*, pp. 986–995, 2017.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Chiyu Ma, Jon Donnelly, Wenjun Liu, Soroush Vosoughi, Cynthia Rudin, and Chaofan Chen. Interpretable image classification with adaptive prototype-based vision transformers. *Advances in Neural Information Processing Systems*, 37:41447–41493, 2024.

Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. *Advances in Neural Information Processing Systems*, 35:17359–17372, 2022.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

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

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin A Raffel, Leandro Von Werra, Thomas Wolf, et al. The FineWeb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

Ofir Press and Lior Wolf. Using the output embedding to improve language models. In Mirella Lapata, Phil Blunsom, and Alexander Koller (eds.), *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 157–163, Valencia, Spain, April 2017. Association for Computational Linguistics. URL https://aclanthology.org/E17-2025/.

Daking Rai, Yilun Zhou, Shi Feng, Abulhair Saparov, and Ziyu Yao. A practical review of mechanistic interpretability for transformer-based language models. *arXiv preprint arXiv:2407.02646*, 2024.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3982–3992, 2019.

Dawid Rymarczyk, Łukasz Struski, Michał Górszczak, Koryna Lewandowska, Jacek Tabor, and Bartosz Zieliński. Interpretable image classification with differentiable prototypes assignment. In *European Conference on Computer Vision*, pp. 351–368. Springer, 2022.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

Noam Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020. URL https://arxiv.org/abs/2002.05202.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. *arXiv preprint arXiv:1905.05950*, 2019.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a. URL https://arxiv.org/abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b. URL https://arxiv.org/abs/2307.09288.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965, 2023.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: A circuit for indirect object identification in GPT-2 small. *arXiv preprint arXiv:2211.00593*, 2022.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *arXiv preprint arXiv:2002.10957*, 2020.

Bowen Wei and Ziwei Zhu. Protolens: Advancing prototype learning for fine-grained interpretability in text classification. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4503–4523, 2025.

Ximing Wen and Rezvaneh Rezapour. A transformer and prototype-based interpretable model for contextual sarcasm detection. *arXiv preprint arXiv:2503.11838*, 2025.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *Advances in Neural Information Processing Systems*, 37:115491–115522, 2024.

Chris Ying, Sameer Kumar, Dehao Chen, Tao Wang, and Youlong Cheng. Image classification at supercomputer scale. In *Proceedings of the Systems for Machine Learning Workshop at NeurIPS 2018*, 2018. URL https://learningsys.org/nips18/assets/papers/84CameraReadySubmissionYing_Kumar_Supercomputer.pdf.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems 32*, Vancouver, Canada, 2019. doi: 10.5555/3454287.3455397. URL https://openreview.net/references/pdf?id=S1qBAf6rr.

Fred Zhang and Neel Nanda. Towards best practices of activation patching in language models: Metrics and methods. *arXiv preprint arXiv:2309.16042*, 2023.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, pp. 649–657, 2015.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. Siren's song in the AI ocean: A survey on hallucination in large language models. *Computational Linguistics*, pp. 1–46, 2025.

Rosie Zhao, Depen Morwani, David Brandfonbrener, Nikhil Vyas, and Sham Kakade. Deconstructing what makes a good optimizer for language models. *arXiv preprint arXiv:2407.07972*, 2024.

A Appendix

A.1 AI Usage

We have used LLMs for proofreading the paper and to polish writing, for retrieval and discovery of related work, and for low-level coding help, e.g. to help us produce the prototype interpretability html. We have checked all AI output, and have verified that the resulting code is correct and works as expected.

A.2 Additional Details on Experiment Setup for Long-Context Scalability and Large-Scale Training

Batch size: We have found that batch size of 32 works best for training among 32, 64, 128, for all models. Lower batch size values were not considered to preserve parallelisability and reduce number of training steps. We keep this batch size (32) in larger experiments as well, for simplicity, and only select the learning rate from a handful of scaling options. Furthermore, smaller batch sizes generalize better than large batch sizes even with large-scale data (Masters & Luschi, 2018); large batch sizes are mainly used for hardware utilization and training speed-up as they require fewer steps to finish training (Ying et al., 2018).

Learning rate: The best learning rates found via the automatic hyperparameter search for the default model sizes are: LLaMA: 1.6e-3, Mamba: 3.8e-3, DeltaNet: 6.8e-3, and ProtoT: 2.0e-3.

For the **long-context scalability experiment**, we have tried increasing the learning rate accordingly (by square root of context size ratio), as per AdamW scaling laws (Li et al., 2024), because extended context is computationally-similar to a larger batch size. However, we have found that scaling the learning rate helps only for DeltaNet and only in the large-scale model/data setting. In the results, we report only the best value from scaled vs non-scaled LR for all models.

For the **large-scale training experiment**, we ran each model with the best hyp-s from the hyp search, and with scaled version thereof. We observed instability with Mamba, so we reduced the LR until it reached stability (from 3.8e-4 down to 2.3e-4). For all other models, we report results with the best-found learning rates (above).

A.3 Prototype Intervention Experiments

To move beyond correlational observations, we designed an intervention experiment to probe the functional role of individual prototypes within the model's predictive process. This methodology involves systematically manipulating a single prototype by re-initializing it with random noise, and then measuring the resulting impact on the model's output probabilities for a targeted linguistic task. By quantifying this change, we can assess the prototype's influence and determine its functional importance for a specific prediction.

A.3.1 Identifying and Targeting Concept-Specific Prototypes

To identify prototypes that appear to encode distinct, human-understandable concepts. We analyze the top-activating sentences for each prototype from the visualization reports [need to ref to Matteo's part about this visualization]. Based on this analysis, we selected three prototypes from Layer 9 for our study. The prototype *L9 P7*(Fig. 2), which consistently activates on sentences containing words such as *'women'* and *'girls'*, we hypothesize that *L9 P7 is a key causal component in the model's representation of the 'female' concept*. Similarly, we identified prototype *L9 P18*(Fig. 3) as a representation for the 'male' concept, as it shows high activation for words like *'man'* and *'boy'*. Finally, prototype *L9 P2*(Fig. 4) was selected as a control, as it did not exhibit a clear, gender-coded semantic preference.

Figure 2: Visualization for prototype **L9 P7**

Test Case Construction. To create a controlled and relevant test set, we began with a seed sentence identified from our initial visualization analysis. This sentence was the top-ranked example from the FineWeb test set that maximally activated the 'write' gate of our primary target, prototype L9 P7. To expand our test set while maintaining semantic consistency, we then prompted a large language model (Gemini 2.5 Pro) to generate six additional sentences thematically similar to the seed sentence, each required to contain the keywords 'women' and 'girls'.

The resulting corpus of seven sentences used in our experiments is as follows:

- *"did you know that there is a government strategy for women and girls in sports and active recreation to address the inequalities of girls' and women's"* (seed sentence from FineWeb)
- *"Many organizations are working on programs that focus on empowering women and girls to participate equally in science and technology."*

Figure 3: Visualization for prototype **L9 P18**

Figure 4: Visualization for the control prototype **L9 P2**

- *"Did you know that several global initiatives aim to protect the rights of women and girls from violence and discrimination?"*
- *"Education policies are increasingly emphasizing equal opportunities for women and girls to excel in leadership roles."*
- *"Access to healthcare remains a critical issue, and governments are creating strategies to improve services for women and girls."*
- *"International campaigns highlight how climate change disproportionately affects women and girls in vulnerable communities."*
- *"Did you know that mentorship networks are being created to support women and girls in pursuing careers in engineering and mathematics?"*

From this corpus, we defined our test cases. Each case consists of a context (the sentence preceding a target word) and a completion token (the target word itself). For this study, we focused on the probability of the target completions 'women' and 'girls'.

Results: After establishing a baseline probability for each test case using the unmodified model, we create a copy of the model for each intervention. The intervention method used is Disruption, where the parameter vector of the target prototype (L9 P7, L9 P18, or L9 P2) is re-initialized with

random noise, scaled according to the model's original initialization scheme. This procedure erases the prototype's learned knowledge while preserving the overall model architecture. We then measure the post-intervention probability of the completion token.

The results of our intervention experiments are summarized in Table 6. To focus the analysis on contexts where the target word is considered a plausible completion by the model, we excluded test cases where the baseline probability of the target completion was below 1%.

Table 6: Results of disrupting prototypes L9 P7, L9 P18, and L9 P2. The table shows the relative percentage change in the probability of the target completion word ('women' or 'girls') compared to the baseline.

		Relative Change in Probability (%) After Intervention		
Context Sentence (Truncated)	*Baseline Prob. (%)*	*L9 P7 ('female')*	*L9 P18 ('male')*	*L9 P2 (Control)*
Target Completion: 'women'				
...inequalities of girls' and women's	3.21	**-17.80%**	+11.50%	+0.74%
...empowering women and girls to participate...	4.24	-3.00%	-0.13%	-0.17%
...protect the rights of women and girls...	13.54	+1.37%	+1.43%	+0.09%
...equal opportunities for women and girls...	10.14	-0.67%	-0.31%	-0.75%
...climate change disproportionately affects women...	11.87	+1.81%	+0.12%	+0.34%
Target Completion: 'girls'				
...inequalities of girls' and women's	2.80	**-10.62%**	+0.50%	+0.03%
...empowering women and girls to participate...	68.55	+0.11%	+0.28%	-0.28%
...protect the rights of women and girls...	78.63	-0.45%	+0.64%	-0.04%
...equal opportunities for women and girls...	60.49	-0.17%	+0.56%	-0.19%
...improve services for women and girls.	64.33	-1.56%	+0.62%	-0.15%
...climate change disproportionately affects women...	68.66	-1.01%	+1.39%	-0.10%
...support women and girls in pursuing careers...	38.32	-3.89%	+2.39%	-0.55%

Our results reveals a clear causal link between prototype *L9 P7* and the model's representation of female-coded concepts. Disrupting this 'female' prototype significantly *decreased* the probability of target words like 'women' (-17.80%) and 'girls' (-10.62%), particularly in less constrained contexts. This effect, however, diminished in test cases where the baseline probability was already very high (e.g., $> 60\%$), suggesting that highly predictable completions are more robust and less reliant on any single prototype. The specificity of this function was confirmed by a control experiment where disrupting an unrelated prototype, *L9 P2*, yielded only negligible changes, proving our findings are not artifacts of random model perturbations. Furthermore, the interventions uncovered a more sophisticated dynamic: disrupting the 'male' prototype, *L9 P18*, consistently *increased* the probability of female-coded words. This suggests an inhibitory or competitive relationship, where the model refines its predictions by balancing between opposing semantic concepts. Taken together, these results demonstrate that the model utilizes specific, functionally distinct, and interacting prototypes to represent and manipulate complex concepts like gender.

A.4 Downstream (Details)

We provide the training protocol and hyperparameter configuration used for the GLUE downstream experiments, covering datasets and splits, preprocessing, optimization, early-stopping/selection on dev, and the hyperparameter sweep and choice rules.

Training protocol: We evaluate four language model architectures: ProtoT, LLaMA, Mamba, and DeltaNet, on the GLUE benchmark under a unified experimental protocol to ensure fair comparison. Unless stated otherwise, all models use the same tokenizer and preprocessing, namely a BPE tokenizer trained on FineWeb with a 16K vocabulary. Inputs are formed as single-sentence or sentence-pair prompts according to the task, with a maximum sequence length of 512. To avoid leakage, we fine-tune on the official training split, select hyperparameters and checkpoints on the official development split (dev) using early stopping We follow the official GLUE metrics: accuracy for SST-2, QNLI, MNLI, QQP, RTE, and WNLI (or the primary metric reported by the official script), the accuracy and F1 pair for MRPC and QQP, Matthews correlation for CoLA, and Pearson and Spearman correlations for STS-B.

Optimization and regularization are aligned across models. We use the AdamW optimizer together with a linear learning-rate schedule with warmup. We apply selective weight decay consistent with pre-training: decay is applied to affine weights that benefit from it, while embeddings, normalization layers, and biases receive no decay. The weight-decay coefficient is 0.01. The batch size is 16. Fine-

tuning runs for up to 3 epochs with early stopping on dev, and the dev-best checkpoint is used to generate test predictions. Unless otherwise specified, a fixed random seed is used across tasks and models to support reproducibility.

Hyperparameter selection: Because architectures differ in optimization sensitivity, we conduct per-model hyperparameter selection. For each model we run small grid searches on two representative tasks, SST-2 (medium-scale binary classification) and MNLI (large-scale multi-class classification). We sweep learning rates over a logarithmic grid that includes 2.5e-5, 3.5e-5, 5.5e-5, 1e-4, 2e-4, 3e-4, 4e-4, 5e-4, 7e-4, 8.5e-4, 1e-3, and we sweep warmup ratios over 6% and 10%. The best learning rate and warmup found per model on these representative tasks are then fixed for that model across the remaining GLUE tasks. All other training details, such as batch size, maximum length, selective decay, optimizer settings, and early-stopping criterion, remain identical across models.

The final per-model settings in our environment are as follows. PrototypeAttn uses a learning rate of 3.5e-5 with 6% warmup. LLaMA uses a learning rate of 5.5e-5 with 10% warmup. Mamba uses a learning rate of 1e-4 with 10% warmup. DeltaNet uses a learning rate of 7e-4 with 10% warmup.

A.5 Robustness (Details)

This appendix details the perturbation set, metrics, and slice-level statistics for the black-box robustness experiments.

A.5.1 Perturbation Dataset Construction

We created a dedicated perturbation dataset (`perturbation_benchmark_clean.jsonl`) spanning seven categories of *meaning-preserving* surface noise, 500 pairs each (3,500 total): *Synonyms* (WordNet (Miller, 1995), filtered by semantic similarity and lexical heuristics), *Typos* (single-character keyboard noise), *Spelling variants* (AmE vs. BrE; e.g., *color* $\rightarrow$ *colour*), *Morphological variants* (e.g., singular $\rightarrow$ plural), *Contractions/Expansions* (e.g., *do not* $\leftrightarrow$ *don't*), *Punctuation/Casing* (insertion/removal), and *Abbreviations/Short forms* (e.g., *Doctor* $\rightarrow$ *Dr.*). Pairs were generated via rule-based perturbations, curated resources (abbreviation/contraction pools), and WordNet substitutions, then filtered with a two-stage pipeline: (i) Sentence-BERT similarity (Reimers & Gurevych, 2019) (MiniLM (Wang et al., 2020)), and (ii) lexical heuristics (frequency/casing). Source corpora: WikiText-2 (Merity et al., 2016), DailyDialog (Li et al., 2017), and AG News Zhang et al. (2015).

Table 7: Example sentence pairs from the perturbation benchmark.

Category	**Original**	**Perturbed**
Abbreviation	*Doctor Smith arrived.*	*Dr. Smith arrived.*
Contraction	*I cannot go.*	*I can't go.*
Synonym	*He was happy.*	*He was glad.*
Spelling	*I like this color.*	*I like this colour.*

A.5.2 Variance Statistics

To quantify dataset quality, we report variance in terms of (i) semantic similarity (cosine similarity of MiniLM embeddings) and (ii) character-level edit distance (Levenshtein, 1966). Table 8 summarizes per-slice averages. We observe that some slices produce small lexical changes but potentially large distributional effects (e.g., typos), while others involve more extensive edits but maintain high semantic similarity (e.g., abbreviations, synonyms), confirming that the dataset spans a wide spectrum of perturbation difficulty.

A.6 Throughput Benchmarks (ProtoT, Mamba, LLaMA, DeltaNet)

We evaluate under identical conditions: same data pipeline, optimizer, precision (BF16), sequence length 256, and batch sizes 32 and 128. FLOP counts are per-sample (forward+backward) where

Table 8: Variance statistics for the perturbation benchmark (3,500 pairs total). Best values in bold.

	Synonym	**Typo**	**Spelling**	**Morphology**	**Contraction**	**Punctuation**	**Abbreviation**
Avg. Similarity	0.828	0.775	0.956	0.881	0.895	0.983	0.894
Avg. Edit Distance	5.89	1.03	1.20	1.00	2.54	1.09	7.23

Table 9: Training throughput (it/s; higher is better) and elapsed time (s; lower is better) for matched-depth/width models at seq. len. 256 (BF16). FLOPs are reported in units of $\times 10^5$ (forward+backward). *When compilation was unavailable, values reflect the fastest steady-state runs without compilation.*

Model	**Batch**	**it/s**	**Elapsed (s)**	**FLOPs/sample** ($\times 10^5$)	**Total FLOPs** ($\times 10^5$)	**Params**
ProtoT	32	25.2	34.57	41,583.0	1,330,657.0	12,205,266
ProtoT	128	7.6	31.32	41,583.0	5,322,625.2	12,205,266
Mamba	32	11.9	58.17	34,734.9	1,111,517.4	6,724,352
Mamba	128	3.2	54.26	34,734.9	4,446,069.4	6,724,352
DeltaNet	32	3.5	222.88	—	—	12,963,456
DeltaNet	128	1.8	182.06	—	—	12,963,456
LLaMA	32	**55.1**	26.16	49,341.5	1,578,929.0	**12,938,496**
LLaMA	128	23.6	**22.30**	49,341.5	6,315,714.3	12,938,496

obtainable. *Observations*: Table 9 summarizes training throughput at batch sizes 32 and 128 for matched-depth/width models. LLaMA attains the highest throughput overall (**55.1** and **23.6** it/s). ProtoT sustains **25.2** and **7.6** it/s and is $\sim$ 2.1–2.4$\times$ faster than Mamba (11.9 and 3.2 it/s) at the same backbone. The FLA-based DeltaNet baseline, evaluated without fused delta kernels and with `torch.compile` disabled, reaches 3.5 and 1.8 it/s (batch 32/128).

A.7 Ablations

A.7.1 Layer-0 Routing Ablations

We ablate the three mitigations that stabilise the layer-0 router: (i) sharing the write/read routing distribution, (ii) sharpening the initial temperature ($\tau_0 = 3.0$), and (iii) adding a $k = 5$ depth-wise convolution to the write-value path of layers 0–1. Each configuration fine-tunes a 6-layer ProtoT on the FineWeb 18k/4k split (sequence length 256, seed 0) for three epochs, using the same optimiser, tokenizer, and learning rate as the main experiments. We report best validation perplexity alongside routing diagnostics logged on the dev set.

Table 10: Layer-0 routing ablations on FineWeb. Metrics come from the final validation epoch (`val_router_stats.csv`) and the best dev perplexity tracked during training. Lower perplexity, Gini, and top-1 probability imply healthier routing; higher $\bar{\alpha}_0$ indicates the ReZero gate remains active. Best values are in bold.

Variant	**Shared** L_0	$\boldsymbol{\tau_0}$ **init**	**Write conv**	**Best val ppl** $\downarrow$	$\bar{\boldsymbol{\alpha}}_{\mathbf{0}}$ $\uparrow$	**Gini** $\downarrow$	**top-1** $\downarrow$
All mitigations (baseline)	On	3.0	$k = 5$	**133.3**	**0.672**	**0.034**	**0.079**
No shared routing	Off	3.0	$k = 5$	133.4	0.658	0.064	0.082
τ reset to 1.0	On	1.0	$k = 5$	133.6	0.653	0.035	0.088
No write conv	On	3.0	Off	145.7	0.354	0.097	0.177
All mitigations off	Off	1.0	Off	149.9	0.261	0.243	0.373

The convolution contributes most to stability: removing it roughly doubles router concentration (top-1 rises from 0.079 to 0.177), increases hub inequality, and cuts the layer-0 ReZero gate in half, ultimately worsening perplexity by $+12.4$ points. Shared routing and the sharpened τ_0 have smaller individual effects on perplexity, but together they keep hub utilisation uniform (gini 0.034) while allowing the gate to stay near its baseline value. Disabling every mitigation reproduces the original alpha-collapse, dropping $\bar{\alpha}_0$ to 0.261 and letting a single hub monopolise 37% of the mass.

Interpretation. Shared write/read routing and the sharper initial temperature primarily act as regularisers: they prevent the router from collapsing mass onto a few hubs without hurting sample efficiency. The depth-wise convolution, in contrast, provides an expressivity boost that both improves perplexity and raises the effective signal scale entering layer 0; once it is removed the router cannot maintain broad support and the ReZero gate decays. The combination of all three mitigations therefore offers a balanced trade-off between stability and performance.

A.8 Additional Intepretability examples

In this section we provide additional examples from the write gate activation interpretability experiment, useful to better illustrate results about learned concept representation.

Figure 5: Visualization for prototype **L0 P18**. Half-life = 12.8

Figure 6: Visualization for prototype **L1 P14**. Half-life = 13.2

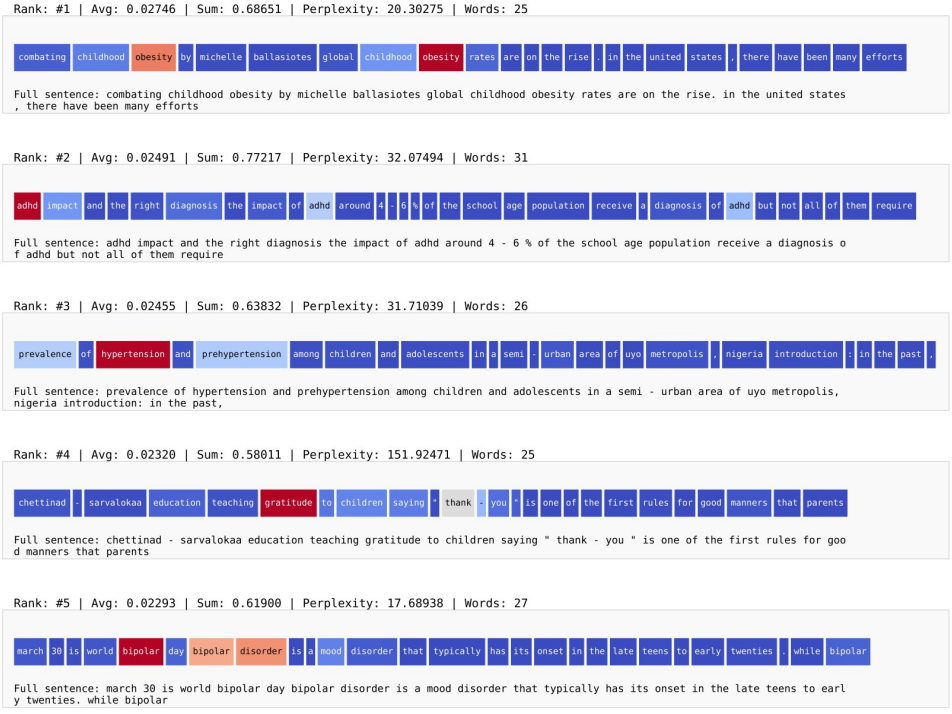

Figure 7: Visualization for prototype **L7 P31**. Half-life = 12.7

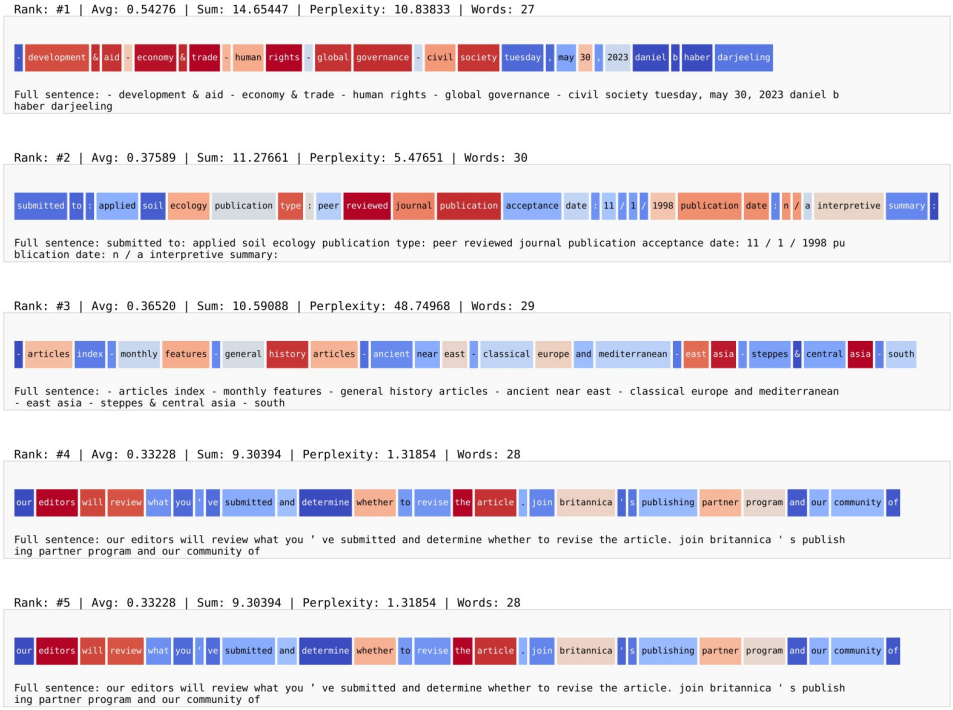

Figure 8: Visualization for prototype **L8 P5**. Half-life = 0.140

Figure 9: Visualization for prototype **L10 P8**. Half-life = 0.510