GROWING PROTOTYPES ON DEMAND: SCALABLE PROTOTYPE LEARNING FOR CAUSAL LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Enabling causal language models to produce interpretable evidence remains challenging. Prototype-based reasoning offers "this looks/reads like that" interpretability, but compressing diverse contextual training evidence into reusable prototypes remains challenging at scale. We propose a non-parametric approach that frames prototype learning as clustering in a frozen model's hidden space using a Dirichlet Process Gaussian Mixture Model (DP-GMM), which allows the number of components to grow only when needed. To scale to millions of hidden states, we introduce a tree-structured DP-GMM that performs coarse-to-fine mixture fitting, with a separate prototype set and prediction head fitted for each task and training distribution on a frozen backbone. The expectations of leaf means are projected to training hidden states to obtain example-grounded prototypes, whose normalised distance similarities drive a learned prediction head. This design enables scalable DP-GMM fitting with node-local cost independent of global prototype capacity for fixed local sample size, hidden dimension, truncation, and fitting settings. Experiments across language modelling, captioning, and VQA fit approximately 2.3M–7.4M hidden states into 5.8k–7.6k prototypes, with small-to-moderate performance drops. Paired interventions show that retaining a small top-matched set largely preserves performance, while removing it severely degrades performance, which supports the computational faithfulness.

1 INTRODUCTION

Causal language and multimodal vision-language models (Vaswani et al., 2017; Grattafiori et al., 2024; Li et al., 2022) have achieved significant performance and are now widely deployed across a wide range of applications, including text generation, image captioning and visual question answering. However, the scale and diversity of their training dataset present challenges for interpretability, which is crucial for trustworthiness and accountability (Doshi-Velez & Kim, 2017; Rudin, 2019; Murdoch et al., 2019). Consequently, there is an increasing interest in improving the interpretability of these models.

A promising approach to enabling inherently interpretable deep models is *prototype learning* (Chen et al., 2019). It leverages prototype layers, where each neuron stores a latent feature vector representing a specific pattern from the training set, known as a prototype. The deep models' predictions are then made by measuring the similarities between features of the current samples and the prototypes. Therefore, prototype learning can offer a "this looks/reads like that" style of interpretability.

However, prototype learning exhibits a natural limitation in scalability: the number of prototypes required tends to grow along with both the dimension of the output layer and the size and diversity of the training dataset. For example, ProtoPNet (Chen et al., 2019) allocates a fixed budget of 10 prototypes per class, resulting in 2,000 prototypes for the CUB-200-2011 bird classification task, which has 200 classes and approximately several thousand training images. In NLP, for relatively simple sentence classification problems, the method proposed by Fanconi et al. (2023) follows a similar design choice, defining 10 prototypes per class for binary Rotten Tomatoes (RT) movie review classification and the four-way AG News dataset. By contrast, for more complex next-token prediction tasks, kNN-LM (Khandelwal et al., 2020) stores contextual representations associated with training tokens, which can result in billions of entries, such as 3 billion tokens for the WIKI-3B corpus. These limitations motivate a compact representation of contextual support.

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

Scaling prototype learning requires compressing diverse contextual training evidence into reusable regions of hidden space. There are recent attempts to reduce the number of required prototypes. The Sparse Neural Editor (He et al., 2020) introduces a latent prototype distribution sampled from a symmetric Dirichlet prior, which encourages a sparse prototype selection distribution over a large pool of prototype candidates (either the full training set or a large subsample of it) when a small concentration parameter is used. However, as the dataset grows, adding more training examples expands the dimensionality of the simplex, forcing the model to distribute probability mass across more components. Meanwhile, using a symmetric Dirichlet prior assumes all candidates are equally likely a priori. Both factors increase the difficulty of learning a meaningful prototype allocation as the dataset size scales.

In this paper, we present a novel methodology that formulates prototype learning as a non-parametric clustering problem in the model's hidden representation space, employing a Dirichlet Process Gaussian Mixture Model (DP-GMM) prior (Ferguson, 1973; Antoniak, 1974; Rasmussen, 1999). The inferred cluster means are projected to training hidden states to construct a prototype layer, enabling example-grounded prediction while keeping the language model backbone fixed. The Dirichlet Process (DP) prior provides a principled non-parametric framework in which the number of active mixture components adapts to the data, subject in our implementation to finite local truncation and tree stopping rules. As the corpus grows, the DP can introduce new components when warranted by the observations, within the tree's capacity ceiling. The concentration parameter, in turn, governs the rate at which novel components are expected to arise, replacing the need for a predefined prior over the importance or number of components.

A remaining challenge is the computational and memory overheads of DP-GMMs. Although the non-parametric nature of the DP in principle permits an unbounded number of prototypes, fitting a single DP-GMM to massive token collections is computationally expensive. First, the computational and memory costs become prohibitive. To approximate the theoretically infinite mixture, the model must adopt a large truncation number K, resulting in inference complexity and memory usage to grow with $T_{\text{total}} \times K$, where T_{total} is the total number of tokens in the corpus. Second, the model suffers from difficulties in termination and capacity selection: under a global mixture, deciding whether to further increase K typically relies on ad-hoc heuristics rather than principled, locally verifiable saturation criteria. Collectively, these limitations render monolithic DP-GMM impractical in large-data regimes and motivate an extension that scales more gracefully.

To mitigate this issue, we propose a tree-structured DP-GMM as a scalable mixture-fitting mechanism. It first performs coarse fitting at the top level with a small truncation, then recursively refines each subset locally, deciding at each node whether to split further or stop. The means of the leaf-node mixture components are projected to their nearest training hidden states to form a prototype layer. Each prototype thus represents a contextual support region grounded in a training example. To make predictions, we compute normalised distances between a current hidden state and these prototypes, which are then aggregated into logits by a prediction head. Throughout prototype fitting and prediction-head training, the backbone language model remains *frozen*, and the prototype head replaces the original prediction head.

This design yields several strengths. (1) **Scalable fitting**: a global DP-GMM fitting is decomposed into many node-local ones that are parallelisable, with significantly reduced local compute and memory footprints. (2) Coarse-to-fine statistical **efficiency**: the model first learns coarse-grained clusters and then adaptively increases local complexity, avoiding the waste and early overfitting associated with initialising a large global K. (3) Preservation of **non-parametric flexibility**: although each node uses a small truncation, the total number of leaf clusters can far exceed any single truncation level, which maintains effectively unbounded representational capacity. (4) Well-defined **stopping criteria**: splitting stops at a node when only one mixture component is active or the maximum depth is reached. Tree construction terminates once every leaf satisfies at least one of these conditions.

2 METHODOLOGY

In this section, we formalise our prototype-learning framework for causal language and multimodal vision-language modelling and describe how prototypes are learned directly in the backbone model's hidden representation space. Firstly, we begin with rewriting autoregressive likelihood in terms of token-level hidden states and introducing a prototype-mixture view of prediction (Section 2.1). Secondly, a Dirichlet Process Gaussian Mixture Model (DP-GMM) prior over hidden states is in-

troduced to obtain a sparse, potentially unbounded set of prototypes (Section 2.2). Then, a tree-structured procedure is proposed to scale DP-GMM fitting via coarse-to-fine local mixtures (Section 2.3). Finally, we show how the learned prototypes define an explicit prototype-activation layer and a prediction layer (Section 2.4).

2.1 Prototype Learning for Causal Language Modelling

Causal language modelling. Given a dataset $\mathcal{D} = \{(x^{(n)}, I^{(n)})\}_{n=1:N}$, where each sample consists of a textual sequence $x^{(n)} = x^{(n)}_{1:T_n}$ and an optional non-text modality $I^{(n)}$ (e.g., an image), N is the number of samples, and T_n is the length of sample n. In causal language and vision-language modelling, the data likelihood factorises autoregressively as

$$p(\mathcal{D}) = \prod_{n=1}^{N} \prod_{t=1}^{T_n} p\big(x_t^{(n)} \mid x_{1:t-1}^{(n)}, I^{(n)}\big). \tag{1}$$

Let f_θ denote a backbone network, such as a Transformer, with parameters θ. For each token position (n, t), a hidden state can be computed by

$$h_t^{(n)} = f_\theta\big(x_{1:t-1}^{(n)}, I^{(n)}\big) \in \mathbb{R}^d. \tag{2}$$

Using these hidden states, we can equivalently write the likelihood as:

$$p(\mathcal{D}) = \prod_{n=1}^{N} \prod_{t=1}^{T_n} p\big(x_t^{(n)} \mid h_t^{(n)}\big). \tag{3}$$

Prototype learning. Assume that the model uses a set of M prototypes $\mathcal{C} = \{\mathbf{c}_i\}_{i=1}^{M}$ in the hidden space, with a latent prototype assignment $z_t^{(n)} \in \{1, \dots, M\}$ for each position (n, t). We can write the token distribution as a mixture over prototypes:

$$p\big(x_t^{(n)} \mid h_t^{(n)}\big) = \sum_{z_t^{(n)}=1}^{M} p\big(z_t^{(n)} \mid h_t^{(n)}\big) p\big(x_t^{(n)} \mid h_t^{(n)}, \mathbf{c}_{z_t^{(n)}}\big), \tag{4}$$

where $p\big(z_t^{(n)} \mid h_t^{(n)}\big)$ measures how well the hidden state $h_t^{(n)}$ matches prototype $\mathbf{c}_{z_t^{(n)}}$. This corresponds to a soft mixing of prototypes.

2.2 DP-GMM over Hidden States

Our goal is to learn a potentially unbounded but sparse set of prototypes directly within the language model's hidden space. To this end, we place a Dirichlet Process Gaussian Mixture Model (DP-GMM) prior over the hidden states and interpret each mixture component as a prototype.

Prior and generative model. We use the stick-breaking construction of the Dirichlet Process. Under this construction, an infinite sequence of latent variables $\{\beta_k\}_{k=1}^{\infty}$ is drawn independently from a Beta distribution: $\beta_k \sim \mathrm{Beta}(1, \alpha)$, where $\alpha > 0$ denotes the concentration parameter of the DP. These variables define the mixture weights $\{\pi_k\}_{k=1}^{\infty}$ via

$$\pi_k = \beta_k \prod_{i<k} (1 - \beta_i), \quad k = 1, 2, \dots. \tag{5}$$

For each mixture component k, we draw its parameters independently from a base distribution $(\mu_k, \Sigma_k) \sim G_0$, where G_0 is a Normal-Inverse-Wishart (NIW) prior:

$$\mu_k \mid \Sigma_k \sim \mathcal{N}(\mu_0, \Sigma_k/\kappa_0), \qquad \Sigma_k \sim \text{Inverse-Wishart}(\nu_0, \Psi_0). \tag{6}$$

Each pair (μ_k, Σ_k) defines a Gaussian component in the hidden space and μ_k plays the role of a prototype. Given $(\pi, \{(\mu_k, \Sigma_k)\}_k)$, the generative model for the hidden states $h_t^{(n)}$ can be formulated as

$$z_t^{(n)} \mid \pi \sim \mathrm{Categorical}(\pi), \qquad h_t^{(n)} \mid z_t^{(n)}, \{(\mu_k, \Sigma_k)\}_{k=1}^{\infty} \sim \mathcal{N}\big(\mu_{z_t^{(n)}}, \Sigma_{z_t^{(n)}}\big). \tag{7}$$

Marginalizing out the latent assignments $z_t^{(n)}$ yields the standard mixture likelihood for the hidden states:

$$p\big(H \mid \pi, \{(\mu_k, \Sigma_k)\}_k\big) = \prod_{n=1}^{N} \prod_{t=1}^{T_n} \sum_{k=1}^{\infty} \pi_k \, \mathcal{N}\big(h_t^{(n)}; \mu_k, \Sigma_k\big), \tag{8}$$

Algorithm 1 Hierarchical Learning of Tree-structured Truncated DP-GMMs

Require: Dataset $H = \{h_n\}_{n=1}^{N_H}$, truncation K, max depth $d_{\max}$, DP concentration α
Ensure: Tree $T = (V, E)$ with node-local DP-GMMs

```
Create root node r with I_r = {1, ..., N_H} and depth(r) = 0
V ← {r}, E ← ∅
Initialize a queue Q ← {r}
while Q is not empty do
    Pop node v from Q
    Fit a truncated DP-GMM with K components on {h_n : n ∈ I_v}, obtaining mixture weights
    π_v, component parameters {(μ_{v,k}, Σ_{v,k})}_{k=1}^K, and assignments {z_{n,v}}_{n∈I_v}
    Compute effective counts m_{v,k} ← |{n ∈ I_v : z_{n,v} = k}| for k = 1, ..., K
    Compute active components A_v ← {k ∈ {1, ..., K} : m_{v,k} > 0}
    K_act(v) ← |A_v|
    if v satisfies either stopping condition in Eq. 16 then
        Mark v as a leaf node
        continue
    end if
    for each k ∈ A_v do
        Create child node u
        I_u ← {n ∈ I_v : z_{n,v} = k}
        depth(u) ← depth(v) + 1
        V ← V ∪ {u},    E ← E ∪ {(v, u)}
        Push u into Q
    end for
end while
```

where $H = \{h_t^{(n)}\}_{t,n}$. In practice, the DP-GMM induces a sparse clustering, where only a finite number of mixture components (i.e., prototypes) are used for any finite dataset. Therefore, we can safely adopt a truncated stick-breaking approximation with K components, restricting the summation in Eq. 8 to $k \in \{1, \dots, K\}$.

Variational inference. We fit the truncated DP-GMM using mean-field variational inference. Specifically, we approximate the posterior with a factorised variational family over latent assignments, stick-breaking variables, and component parameters:

$$q\big(Z, \{\beta_k\}_k, \{(\mu_k, \Sigma_k)\}_k\big) = \prod_{t,n} q(z_t^{(n)}) \prod_{k=1}^{K} q(\beta_k) \prod_{k=1}^{K} q(\mu_k, \Sigma_k), \tag{9}$$

where $Z = \{z_t^{(n)}\}_{t,n}$. The variational parameters are obtained by maximising the evidence lower bound (ELBO),

$$\mathcal{L}(q) = \mathbb{E}_q\left[\log p(H, Z, \{\beta_k\}_k, \{(\mu_k, \Sigma_k)\}_k)\right] - \mathbb{E}_q\left[\log q(Z, \{\beta_k\}_k, \{(\mu_k, \Sigma_k)\}_k)\right], \tag{10}$$

which is equivalent to minimizing $\mathrm{KL}(q \,\|\, p(\cdot \mid H))$. We maximise Eq. 10 via coordinate-ascent (variational EM). In each iteration, we update responsibilities

$$r_{t,k}^{(n)} \;\propto\; \exp\Big(\mathbb{E}_q[\log \pi_k] + \mathbb{E}_q[\log \mathcal{N}(h_t^{(n)} \mid \mu_k, \Sigma_k)]\Big), \tag{11}$$

and then update the variational posteriors of the stick-breaking weights and the Gaussian component parameters using responsibility-weighted sufficient statistics. In downstream use, we project μ_k onto the nearest latent training text token by

$$\mathbf{c}_i = \operatorname*{argmin}_{h \in H} \|h - \mu_i\|_2. \tag{12}$$

2.3 TREE-STRUCTURED DP-GMM

Let $\{h_t^{(n)}\}_{t,n}$ denote all hidden states produced by the backbone f_θ. We flatten and reindex them as a set $H = \{h_n\}_{n=1}^{N_H}$ with $h_n \in \mathbb{R}^d$, where $N_H = \sum_{n=1}^{N} T_n$ is the total number of token positions across the dataset. We learn a tree-structured collection of local truncated DP-GMMs on H using Algorithm 1.

Tree construction. Each node v in the tree is associated with an index set $\mathcal{I}_v \subseteq \{1, \ldots, N_H\}$ specifying which hidden states reach that node. The root r contains all points, $\mathcal{I}_r = \{1, \ldots, N_H\}$, and has depth $\mathrm{depth}(r) = 0$. We maintain a queue $\mathcal{Q}$ of nodes to process, which is initialised as $\mathcal{Q} = \{r\}$. While the queue is non-empty, we pop a node v and fit a truncated DP-GMM with K components on the subset of hidden states $\{h_n : n \in \mathcal{I}_v\}$, using the concentration parameter α. This yields node-specific mixture weights $\boldsymbol{\pi}_v$, component parameters $\{(\mu_{v,k}, \Sigma_{v,k})\}_{k=1}^{K}$, and posterior responsibilities $\{r_{i,v,k}\}_{i \in \mathcal{I}_v,\, k=1:K}$. To assign hidden states to a node, a hard assignment can be applied by

$$z_{i,v} \;=\; \underset{k \in \{1,\ldots,K\}}{\mathrm{argmax}}\; r_{i,v,k}, \qquad i \in \mathcal{I}_v. \tag{13}$$

From these assignments, the *effective counts* are defined as

$$m_{v,k} \;=\; \big|\{i \in \mathcal{I}_v : z_{i,v} = k\}\big|, \qquad k = 1, \ldots, K, \tag{14}$$

and the set of *active components* is defined as

$$A_v \;=\; \big\{k \in \{1, \ldots, K\} : m_{v,k} > 0\big\}. \tag{15}$$

We use $K_{\text{act}}(v) = |A_v|$ to denote the number of active components at node v.

Stopping conditions. Two conditions determine whether to stop splitting node v (Line 10 of Algorithm 1). We stop when the node reaches the maximum depth $d_{\max}$ or when its mixture has only one active component, i.e., $K_{\text{act}}(v) = 1$. Formally, v becomes a leaf node whenever

$$\mathrm{depth}(v) = d_{\max} \;\textbf{or}\; K_{\text{act}}(v) = 1. \tag{16}$$

If neither stopping condition is met, we expand node v by creating one child node u for each active component $k \in A_v$. The child u inherits a subset of indices corresponding to the points assigned to component k at its parent:

$$\mathcal{I}_u \;=\; \{n \in \mathcal{I}_v : z_{n,v} = k\}, \tag{17}$$

$$\mathrm{depth}(u) = \mathrm{depth}(v) + 1, \tag{18}$$

and is added to the queue $\mathcal{Q}$ for subsequent processing. In this way, the root-level DP-GMM performs a coarse partitioning of the hidden space, and deeper nodes recursively refine these partitions into more specialised clusters.

The final output is a tree $\mathcal{T} = (V, E)$, where each node $v \in V$ carries a local DP-GMM $\big(\boldsymbol{\pi}_v, \{(\mu_{v,k}, \Sigma_{v,k})\}_{k=1}^{K}\big)$ over the hidden states that reach it. Let $V_{\text{Leaf}} \subseteq V$ denote the set of leaf nodes. We form the prototype set by collecting the active components from all leaves

$$\mathcal{C}' \;=\; \big\{\mu_{v,k} : v \in V_{\text{Leaf}},\; k \in A_v\big\}, \tag{19}$$

and applying the projection rule in Eq. 12 to obtain $\mathcal{C}$.

2.4 Prediction with Prototype Activations

We turn the prototypes $\mathcal{C} = \{\mathbf{c}_i\}_{i=1}^{M}$ into an explicit prediction layer that operates on the hidden states $\big\{h_t^{(n)}\big\}_{t,n}$. For each position (n, t) and prototype $\mathbf{c}_i$, we first measure how similar the current context $h_t^{(n)}$ is to the prototype vector $\mathbf{c}_i$. Similarity is measured using the Euclidean distance

$$d_{t,i}^{(n)} = \|h_t^{(n)} - \mathbf{c}_i\|_2. \tag{20}$$

Rather than using these absolute distances, a Softmax function is applied to the negative distance to convert them into normalised probability distributions:

$$s_{t,i}^{(n)} := \frac{\exp\left(-d_{t,i}^{(n)}\right)}{\sum_{i'=1}^{M} \exp\left(-d_{t,i'}^{(n)}\right)}. \tag{21}$$

This ensures that the resulting values are non-negative, sum to one, and increase as the distance decreases. This normalised similarity vector is the central *interpretability bottleneck* of our model. Instead of allowing the hidden state $h_t^{(n)}$ to be mapped directly to vocabulary logits, the model must

first express the current context as a distribution over example-grounded prototypes. The subsequent prediction head therefore operates only through prototype activations, making the selected prototypes part of the computational pathway rather than post-hoc explanations. The stacked prototype activations serve as inputs to the prediction head:

$$s_t^{(n)} = \big(s_{t,1}^{(n)}, s_{t,2}^{(n)}, \ldots, s_{t,M}^{(n)}\big)^\top \in \mathbb{R}^M. \tag{22}$$

Let $\mathcal{V}$ denote the vocabulary, and for each token type $v \in \mathcal{V}$, let $w_v \in \mathbb{R}^M$ and $b_v \in \mathbb{R}$ be the parameters of a prediction head. The unnormalized logit for predicting token v at position (n, t) is

$$\ell_{t,v}^{(n)} = b_v + \sum_{i=1}^{M} w_{v,i}\, s_{t,i}^{(n)} = b_v + w_v^\top s_t^{(n)}. \tag{23}$$

Finally, the token distribution is given by the Softmax over these logits:

$$p\big(x_t^{(n)} = v \mid h_t^{(n)}, \mathcal{C}\big) = \frac{\exp\big(\ell_{t,v}^{(n)}\big)}{\sum_{u \in \mathcal{V}} \exp\big(\ell_{t,u}^{(n)}\big)}. \tag{24}$$

3 Experiments

We conduct experiments on three representative tasks: text generation, image captioning, and visual question answering (VQA), which cover both unimodal pure-language modelling and multimodal vision-language modelling.

3.1 Setup

Backbone models. For text generation, we use LLaMA 3.2 models (Grattafiori et al., 2024) at 1B and 3B parameter scales. For both image captioning and VQA, we use the BLIP (Li et al., 2022) models. Across the experiments, the model backbones are kept frozen, and only the prototypes and the prediction layers are learned.

Datasets. For text generation, we use the WikiText-2 and WikiText-103 datasets (Merity et al., 2017), two Wikipedia-based language-modelling corpora, where WikiText-103 contains the full collection of over 103 million words, while WikiText-2 is a smaller version. For image captioning, we use the Karpathy split (Karpathy & Fei-Fei, 2015) of the COCO dataset (Lin et al., 2014), which contains around 123K images, each annotated with five crowd-sourced human-written sentences, and uses 5,000 images each for validation and testing. For VQA, we use the VQA v2 dataset (Goyal et al., 2017), a balanced extension of VQA (Antol et al., 2015) built on COCO images with free-form questions and 10 human answers per question.

Learning and optimisation. We learn prototypes in the backbone hidden space. For each task, we first run the frozen backbone on the training split to collect hidden states $H = \{h_n\}_{n=1}^{N_H}$. We then fit tree-structured DP-GMMs following Algorithm 1 with local truncation $K = 10$ at each node and maximum depth $d_{\max} = 5$. Following Eq. 16, nodes stop splitting when they reach the maximum depth or when only a single mixture component remains active. The DP concentration parameter is fixed to $\alpha = 1/K = 0.1$. Each leaf mixture mean $\mu_{v,k}$ is projected to its nearest training hidden state to obtain an example-grounded prototype (Eq. 12). As for the prediction layer, it is optimised with gradient descent and the Adam optimiser.

3.2 Dataset Statistics and Prototype Counts

Table 1 summarises the training set statistics used for prototype learning and reports the resulting number of learned prototypes for each dataset–model pair. For each dataset, we compute the vocabulary size after tokenising the text with the task-specific tokeniser of the corresponding model. We also report "coverage", which is defined as the number of unique tokens whose cumulative frequency covers 90% and 99% of all token occurrences in the training set. Due to the much larger scale of WikiText-103, we apply a subsampling. Details are provided in Appendix B. All the experiments are conducted with a single NVIDIA A6000 GPU with 48GB of GPU memory and 128GB of CPU memory.

Across models evaluated on the same dataset, prototype counts vary only marginally with model scale, and larger models do not necessarily require more prototypes. Instead, the number of prototypes is more strongly associated with dataset characteristics, particularly vocabulary size and task

Table 1: Training-set token statistics and the resulting number of learned prototypes.

Modality	Dataset	# Tokens (million)	Vocab. Size	Coverage† 90%	Coverage† 99%	Model	# proto
Language	WikiText-2	2.3	32,422	7,287	23,682	LLaMA 3.2 1B	5,838
						LLaMA 3.2 3B	5,794
	WikiText-103	115.2	61,066	9,935	33,707	LLaMA 3.2 1B	6,433
						LLaMA 3.2 3B	6,796
Vision-Language	COCO Karpathy	7.4	15,547	673	4,594	BLIP $_{\text{ViT-B+CapFilt-L}}$	7,038
						BLIP $_{\text{ViT-L}}$	7,641
	VQA V2*	6.5	17,424	1,155	7,008	BLIP $_{\text{ViT-B}}$	6,403
						BLIP $_{\text{ViT-B+CapFilt-L}}$	6,110

∗: BLIP uses an encoder-decoder architecture for VQA; therefore, for VQA v2, we primarily focus on the answers generated by the decoder and show their statistics. The vocabulary of the full dataset (both questions and answers) contains 18,572 unique tokens. †: the size of vocabulary covering 90%/99% of token occurrences.

Table 2: Pure-language next-token prediction on the WikiText-2 and 103 test set, evaluated by perplexity (↓).

Backbone	WikiText-2 Origin	WikiText-2 Proto	WikiText-103 Origin	WikiText-103 Proto
LLaMA 3.2 1B	9.21	10.90	8.71	9.42
LLaMA 3.2 3B	7.44	7.97	7.21	8.09

Table 3: Image captioning on the COCO Karpathy test set, evaluated by BLEU@4 (↑) and CIDEr (↑).

Backbone	BLEU@4 Origin	BLEU@4 Proto	CIDEr Origin	CIDEr Proto
BLIP $_{\text{ViT-B+CapFilt-L}}$	39.4	37.8	131.4	128.1
BLIP $_{\text{ViT-L}}$	39.7	38.1	133.3	130.5

modality. Within language modelling, the larger-vocabulary WikiText-103 consistently requires more prototypes than WikiText-2, regardless of model size. Notably, despite having smaller vocabularies, the vision–language datasets require comparable or larger prototype sets than the language datasets, suggesting that multimodal grounding increases the diversity of patterns that must be represented.

3.3 Quantitative Evaluation

We evaluate whether replacing the original prediction head (**Origin**) with our prototype-based prediction layer (**Proto**) can introduce interpretability while preserving task performance. The comparison is to measure the cost of imposing an example-grounded prototype bottleneck on an otherwise frozen backbone. Table 2 reports next-token prediction perplexity on the WikiText-2 and WikiText-103 for LLaMA 3.2 1B and 3B backbones. Proto yields slightly higher perplexity than Origin on both datasets, but the differences remain moderate, ranging from about 0.53 to 1.69 points (approximately 7.1%–18.3%) across these settings. For image captioning, Table 3 reports results on COCO Karpathy for BLIP ViT-B+CapFilt-L and ViT-L backbones. We report BLEU@4 and CIDEr. It shows smaller relative gaps: Proto retains about 96% of BLEU@4 and about 97.5%–97.9% of CIDEr compared with Origin. For VQA, Table 4 reports results on the VQA v2 test-standard split for BLIP ViT-B and ViT-B+CapFilt-L. We report VQA accuracy. It shows that Proto retains about 98.0%–98.7% of the original VQA accuracy. Overall, across unimodal language modelling and multimodal vision-language tasks, the prototype bottleneck introduces small-to-moderate performance gaps while enabling an interpretable prediction mechanism.

3.4 Comprehensiveness and Sufficiency

We quantify how much predictions depend on the most similar prototypes by performing interventions on the prototype activations (DeYoung et al., 2020; Jacovi & Goldberg, 2021; Fanconi et al., 2023). For comprehensiveness, we zero out the top-k most similar prototypes to assess how much the explanation contributes to the prediction. This analysis measures the extent to which the prediction depends on the identified explanations. For sufficiency, we take the opposite approach and retain only the top-k most similar prototypes. For stability, the removal and retaining are performed before the Softmax function (Eq. 21), and the Softmax is applied over the remaining prototypes before computing logits.

Table 5 shows that removing the top-5 or top-10 matched prototypes dramatically worsens perplexity (e.g., drops from 9.42 to 375.95 for 1B with top-5 removed), and as more top matched prototypes are removed, the performance continues to drop. This demonstrates that the top similar prototypes play a crucial role in the model's predictions. Conversely, retaining only the top prototypes, the

Table 4: Visual question answering on the VQA V2 test-standard set, evaluated by VQA accuracy (↑).

Backbone	Origin	Proto
BLIP $_{\text{ViT-B}}$	78.17	77.12
BLIP $_{\text{ViT-B+CapFilt-L}}$	78.32	76.76

Table 5: Comprehensiveness and sufficiency analysis based on removing or retaining only the top-matched prototypes, evaluated using the perplexity of LLaMA 3.2 on the WikiText-103 test set.

Back.	Full	Remove		Retain	
		Top-5	Top-10	Top-5	Top-10
1B	9.42	375.95	675.25	10.49	9.78
3B	8.09	399.08	537.32	9.25	8.13

perplexity remains competitive (e.g., 10.49 for 1B and 9.25 for 3B, both with top-5 only). This indicates that a small number of prototypes largely suffices to support predictions, and that the top-matched prototypes are genuinely influential.

3.5 VISUALISATION

We visualise learned prototypes in Fig. 1. The figure provides an illustrative overview of how our prototype-based framework enables interpretability, where a model's prediction for a test instance is explained by matching training samples as prototypes. In each row, the left side shows the test samples and the model outputs, while the right side shows the selected prototypes. The predicted words (tokens) are highlighted. The **top row** illustrates next-token prediction in a purely textual setting. A test context excerpt ends with a phrase indicating that "the first draft was ...", and the model's predicted continuation is the token "completed". The matched prototype is a training paragraph containing a closely analogous structure, which also ends with "the first draft was completed." The alignment between the test context and the prototype provides an interpretable rationale. The **mid-**

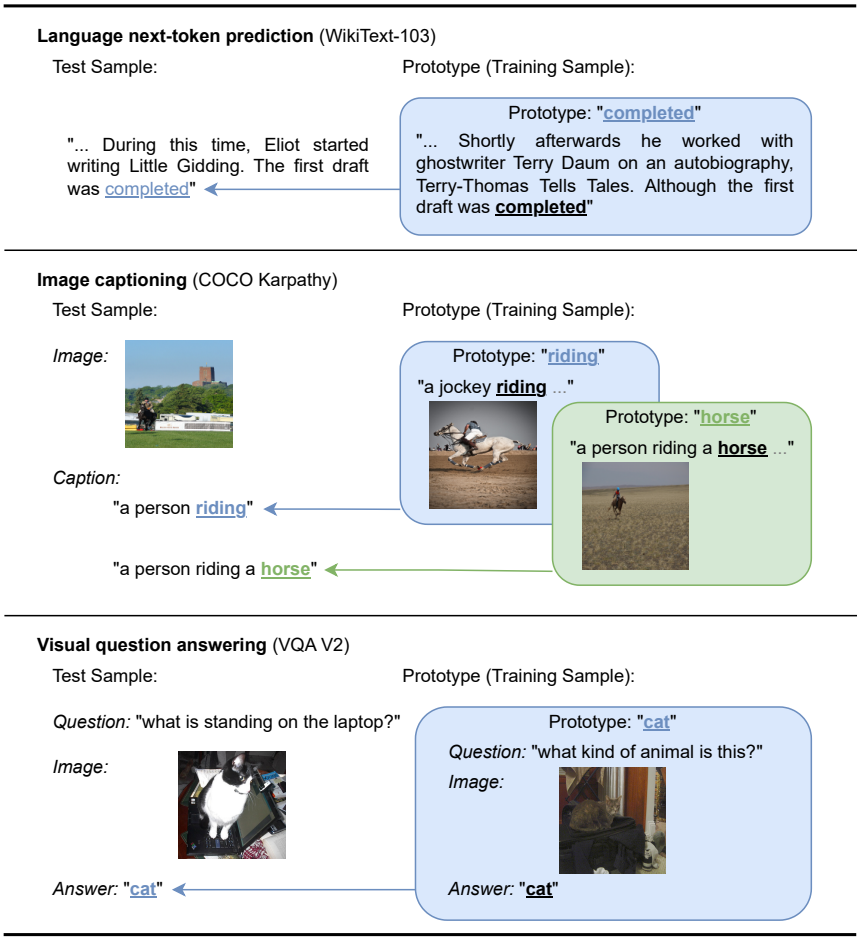


Figure 1: Illustration of prototype-based interpretability across language and vision-language tasks.

Table 6: Evaluation of different tree depths. The number of learned prototypes ("# proto") and the proportion of utilised prototypes relative to the total prototype capacity ("Util.") are reported.

Depth	1B			3B		
	# proto	Util. (%)	Perplexity	# proto	Util. (%)	Perplexity
1	10	100.0	7,033.89	10	100.0	6,679.30
2	100	100.0	1,340.35	100	100.0	1,813.47
3	978	97.8	264.26	1,000	100.0	262.49
4	6,433	64.3	9.42	6,796	68.0	8.09
5	6,433	6.4	9.42	6,796	6.8	8.09
Origin	N/A	N/A	8.71	N/A	N/A	7.21

dle row demonstrates interpretability for image captioning. Two distinct prototypes are shown: one image–caption pair strongly associated with the action word "riding", and another prototype supporting the object noun "horse". The **bottom row** applies the same idea to VQA. The test instance asks, "what is standing on the laptop?" alongside an image showing a cat positioned on a laptop keyboard, and the model answers "cat". The prototype is a training question–image pair that also includes a cat and the same answer. Although the prototype question differs ("what kind of animal is this?"), the shared visual concept (a cat) dominates the test answer, indicating that the model's response is supported by similar visual evidence in the training set. Overall, the figure visualises a unified interpretability strategy across modalities: predictions are explained by the most similar training samples (prototypes).

3.6 Tree Depths

Table 6 evaluates the effect of different tree depths and consequently prototype capacities of the proposed tree-structured DP-GMM. Experiments are conducted on WikiText-103 using LLaMA-3.2 backbones. The utilisation percentage is computed as the ratio between the number of learned prototypes and the maximum number of prototypes allowed at a given depth (e.g., depth 3 allows up to $10^3 = 1{,}000$ prototypes). Models with shallow trees exhibit apparent underfitting. For example, depths 1–3 lead to substantially higher perplexities and nearly 100.0% prototype utilisation, indicating insufficient model capacity. In contrast, a tree depth of 4 achieves low perplexities, which is comparable to those of the original models. Moreover, increasing the tree depth beyond 4 provides no further performance gains and does not increase the number of learned prototypes.

These observations indicate that: (1) the number of learned prototypes (i.e., DP-GMM components) is already sufficient, and (2) the proposed method can grow the number of prototypes on demand, rather than learning too few prototypes (leading to substantially degraded performance) or excessively increasing the number as model capacity grows. Consequently, the method successfully saturates and prevents unnecessary tree expansion and over-parameterisation. Overall, these results empirically support the scalability motivation of our approach and demonstrate its ability to adapt model capacity in a data-driven manner.

4 Conclusion

In this paper, we address the challenge of interpretability in large-scale causal language and multimodal vision-language models by revisiting prototype learning, a promising but poorly scalable approach in high-dimensional and data-rich settings. We reformulate prototype learning as a non-parametric clustering problem in the hidden representation space using a Dirichlet Process Gaussian Mixture Model (DP-GMM), allowing the number of prototypes to be determined by the data rather than fixed a priori. To mitigate the computational and memory limitations of monolithic DP-GMMs, we introduce a tree-structured DP-GMM that enables scalable, coarse-to-fine prototype learning through bounded-width local fits. This design preserves non-parametric flexibility while producing compact, interpretable prototype layers whose elements can be traced back to training samples, while keeping the backbone model frozen. Our results show that: (1) the learned prototype sets remain compact (approximately 5.8k–7.6k prototypes fitted to 2.3M–7.4M hidden states), (2) model quality incurs small-to-moderate degradation relative to the original heads, and (3) paired interventions showing that retaining a small top-matched prototype set largely preserves performance, while removing it severely degrades performance.

AI USE STATEMENT

In this work, we used generative AI tools for grammar check. We have not used generative AI tools for other purposes. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pp. 2425–2433, 2015.

Charles E Antoniak. Mixtures of dirichlet processes with applications to bayesian nonparametric problems. *The annals of statistics*, pp. 1152–1174, 1974.

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.

Anubrata Das, Chitrank Gupta, Venelin Kovatchev, Matthew Lease, and Junyi Jessy Li. Prototex: Explaining model decisions with prototype tensors. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2986–2997, 2022.

Jay DeYoung, Sarthak Jain, Nazneen Fatema Rajani, Eric Lehman, Caiming Xiong, Richard Socher, and Byron C Wallace. Eraser: A benchmark to evaluate rationalized nlp models. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 4443–4458, 2020.

Xingjian Diao, Weiyi Wu, Keyi Kong, Peijun Qing, Xinwen Xu, Ming Cheng, Soroush Vosoughi, and Jiang Gui. Protovqa: An adaptable prototypical framework for explainable fine-grained visual question answering. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 1040–1057, 2025.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

Claudio Fanconi, Moritz Vandenhirtz, Severin Husmann, and Julia Vogt. This reads like that: deep learning for interpretable natural language processing. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 14067–14076, 2023.

Thomas S Ferguson. A bayesian analysis of some nonparametric problems. *The annals of statistics*, pp. 209–230, 1973.

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6904–6913, 2017.

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.

Junxian He, Taylor Berg-Kirkpatrick, and Graham Neubig. Learning sparse prototypes for text generation. *Advances in neural information processing systems*, 33:14724–14735, 2020.

Junxian He, Graham Neubig, and Taylor Berg-Kirkpatrick. Efficient nearest neighbor language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 5703–5714, 2021.

Ta Duc Huy, Sen Kim Tran, Phan Nguyen, Nguyen Hoang Tran, Tran Bao Sam, Anton Van Den Hengel, Zhibin Liao, Johan W Verjans, Minh-Son To, and Vu Minh Hieu Phan. Interactive medical image analysis with concept-based similarity reasoning. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 30797–30806, 2025.

Alon Jacovi and Yoav Goldberg. Aligning faithful interpretations with their social attribution. *Transactions of the Association for Computational Linguistics*, 9:294–310, 2021.

Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3128–3137, 2015.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations*, 2020.

Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International conference on machine learning*, pp. 12888–12900. PMLR, 2022.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Yue Liu, Ke Liang, Jun Xia, Meng Liu, Xihong Yang, Xinwang Liu, Sihang Zhou, and Stan Z Li. Skip: A prototype-based scalable knowledge graph representation learning method. *IEEE Transactions on Neural Networks and Learning Systems*, 2025.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.

Yao Ming, Panpan Xu, Huamin Qu, and Liu Ren. Interpretable and steerable sequence learning via prototypes. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 903–913, 2019.

W James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080, 2019.

Meike Nauta, Ron Van Bree, and Christin Seifert. Neural prototype trees for interpretable fine-grained image recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14933–14943, 2021.

Carl Rasmussen. The infinite gaussian mixture model. *Advances in neural information processing systems*, 12, 1999.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence*, 1(5):206–215, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Eric Wallace, Shi Feng, and Jordan Boyd-Graber. Interpreting neural networks with nearest neighbors. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 136–144, 2018.

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.

Frank F Xu, Uri Alon, and Graham Neubig. Why do nearest neighbor language models work? In *International Conference on Machine Learning*, pp. 38325–38341. PMLR, 2023.

A Related Works

A.1 Prototype Learning

Prototype learning for vision classification. ProtoPNet (Chen et al., 2019) frames image classification as evidence aggregation over similarities between an input's latent parts and learned prototypical parts, producing "this looks like that" explanations grounded in specific regions. Building on this part-prototype notion, ProtoTree (Nauta et al., 2021) replaces additive evidence pooling with a global decision structure, which routes examples through a prototype-parameterised binary tree to provide both a human-readable model and path-based local explanations.

Prototype and example-based reasoning in NLP. Wallace et al. (2018) interpret neural text classifiers by modifying test-time behaviour via deep k-nearest neighbours (kNN) retrieval in representation space, using neighbours to produce more robust uncertainty and feature-importance explanations. This neighbour-based view is conceptually aligned with prototype learning: instead of learned visual parts, explanation arises from similarity to stored examples in an embedding space. Ming et al. (2019) propose ProSeNet, an interpretable and steerable sequence model whose predictions are obtained by comparing inputs with a small set of exemplar prototypes, supporting case-based explanations and expert-driven prototype refinement. Das et al. (2022) introduce ProtoTEx, a white-box text classification architecture in which prototype tensors represent latent clusters of training examples and decisions are explained through distances to influential prototypes and their nearest training instances. Fanconi et al. (2023) more directly transfer the ProtoPNet philosophy to text classification by learning prototype similarities in sentence-embedding space and extracting prediction-relevant words from both the input and matched prototypes.

A.2 Retrieval- and Prototype-based Text Generation

Moving from classification to generation, kNN-LM (Khandelwal et al., 2020) operationalises the same representation-space retrieval idea for language modelling by interpolating a neural LM with a kNN distribution over a datastore, improving long-tail prediction and enabling domain adaptation by swapping the datastore. He et al. (2021) investigate efficient nearest-neighbour language models, showing that retrieval overhead can be reduced substantially while maintaining comparable performance. Xu et al. (2023) analyse why nearest-neighbour language models work, identifying factors such as representation choice, approximate search, and softmax temperature as important contributors to their gains. These studies clarify the memory, latency, and behavioural properties of token-level retrieval, whereas our method can be viewed as a scalable and interpretable alternative based on adaptive prototype structure rather than dense token-level nearest-neighbour lookup. He et al. (2020) adopt a prototype-driven "retrieve a prototype then generate/edit" perspective, but improve efficiency by learning a sparse prototype support set and an amortised retrieval function instead of indexing the full corpus.

A.3 Bayesian Nonparametric

Bayesian nonparametrics provides a classical statistical perspective on models whose effective complexity can grow with the data. The Dirichlet process was introduced by Ferguson (1973) as a prior over random probability measures for Bayesian nonparametric inference. Antoniak (1974) extended this framework to mixtures of Dirichlet processes, providing a foundation for mixture models in which the effective number of components is not fixed in advance. Rasmussen (1999) developed the infinite Gaussian mixture model, showing how Bayesian mixture models can perform inference over an unbounded set of clusters.

A.4 Other Relevant Topics

Recent work has extended prototype- and concept-grounded reasoning to safety-critical and multimodal vision settings. Huy et al. (2025) propose Concept-based Similarity Reasoning (CSR) for classification-based medical image analysis, combining concept-level semantics with patch-level prototype comparison and doctor-in-the-loop spatial interaction. Diao et al. (2025) introduce ProtoVQA, which learns question-aware prototypes and matches them to spatially coherent image patches, allowing classification-based VQA to be traced to question-conditioned visual evidence. In NLP, Wei & Zhu (2025) propose ProtoLens for fine-grained text classification interpretability. ProtoLens extracts prototype-specific sub-sentence spans and aligns learned prototypes with human-understandable training examples, improving the granularity of text explanations. This reinforces the value of localising prototype evidence in language inputs, although the task remains classification rather than open-ended generation. Prototype ideas have also been applied to structured data. Liu et al. (2025) propose SKIP, a scalable knowledge graph representation learning method that uses prototype information to select representative entity anchors. Unlike explanation-focused prototype models, SKIP uses prototypes mainly to improve scalability by summarising large knowledge graphs with compact, representative anchors.

B WikiText-103 Subsampling

WikiText-103 is extremely larger than the other datasets used in the experiments, including 115.2 million tokens (approximately 15-50$\times$ larger). Consequently, using all token-level hidden states for

the DP-GMM is prohibitively memory-intensive. We estimate the total memory required for hidden states as

$$\text{TotalSize} = \#\,\text{Tokens} \times D_{\text{model}} \times b_{\text{dtype}} \times 1024^{-3} \quad (\text{GiB}) \tag{25}$$

where D_{model} is the model hidden dimension, and b_{dtype} is the number of bytes per hidden-state element determined by the datatype (e.g., $b_{\text{dtype}} = 8$ for `float64`). For numerical stability in DP-GMM fitting, we use double precision `float64`. Using LLaMA models with a hidden dimension of 2,048, the full WikiText-103 hidden states would require approximately 1,757.3 GiB of memory solely for data storage.

To make experiments feasible while preserving coverage and diversity, we construct a balanced subsample of WikiText-103. The procedure is as follows. (1) We first run the model over the full corpus and save all token-level hidden states to disk (instead of saving all in memory). (2) To ensure vocabulary coverage, we select one initial hidden state at random for each token type in the vocabulary. (3) For each token type, we then expand its selected set using a furthest point sampling (FPS) strategy: at each iteration, we add the hidden state that maximises the distance to the current selected set for that token, and repeat until the global budget is exhausted. Tokens with fewer occurrences do not contribute further candidates after being fully included (no resampling). This combination of per-type initialisation and FPS encourages both broad token coverage and intra-token representational diversity, while avoiding domination by high-frequency tokens.

We use a total budget of 2.5 million, which is similar to the number of tokens in WikiText-2. After subsampling, the memory requirement for WikiText-103 hidden states is reduced to 38.2 GiB. Table 7 summarises the hidden-state sizes for all datasets used in our experiments.

Table 7: Estimated storage/memory required for token-level hidden states across datasets. For WikiText-103, we report both the full corpus and our balanced subsample ("WikiText-103 (Sub)").

Dataset	# Tokens (million)	D_{model}	Data Size (GiB)
WikiText-2	2.3	2,048	35.5
WikiText-103	115.2	2,048	1,757.3
WikiText-103 (Sub)	2.5	2,048	38.2
COCO	7.4	768	42.3
VQA V2	6.5	768	37.0

C Memory Footprint Analysis for DP-GMM

We further analyse the full memory footprint of fitting a DP-GMM with the collected hidden states. Let K denote the number of mixture components, N_H the number of samples (i.e., "# Tokens"), and D_{model} the dimensionality of the hidden states. The input hidden-state matrix is $H \in \mathbb{R}^{N_H \times D_{\text{model}}}$.

In our tree-structured approach, a DP-GMM is fitted independently at each node. The peak memory usage occurs at the root node, where all hidden states are processed together. In our experiments, each node-level DP-GMM uses $K = 10$. Since the tree construction terminates automatically at depth 4 (as shown in Table 6), the procedure yields at most 10,000 leaf mixture components in total (across all leaves). To provide a comparable baseline against fitting a single model, we therefore consider a single DP-GMM with $K = 10,000$ components.

We approximate the dominant memory usage by accounting for four principal terms in DP-GMM: (i) the input hidden states $H \in \mathbb{R}^{N_H \times D_{\text{model}}}$, which require

$$M_H = b_{\text{dtype}} \times N_H \times D_{\text{model}}; \tag{26}$$

(ii) the responsibility matrix $R \in \mathbb{R}^{N_H \times K}$ storing posterior assignment probabilities, which requires

$$M_R = b_{\text{dtype}} \times N_H \times K; \tag{27}$$

(iii) the component covariance parameters $\Sigma \in \mathbb{R}^{K \times D_{\text{model}} \times D_{\text{model}}}$, which require

$$M_\Sigma = b_{\text{dtype}} \times K \times (D_{\text{model}})^2; \tag{28}$$

and (iv) the second-moment sufficient statistics $S^{(2)} \in \mathbb{R}^{K \times D_{\text{model}} \times D_{\text{model}}}$, which require

$$M_{S^{(2)}} = b_{\text{dtype}} \times K \times (D_{\text{model}})^2. \tag{29}$$

Summing these contributions yields the dominant term of the memory footprint (up to constant-factor working buffers and implementation-specific overheads):

$$M_{\text{total}} \approx b_{\text{dtype}} \left(N_H D_{\text{model}} + N_H K + 2K(D_{\text{model}})^2 \right). \tag{30}$$

In our implementation, we use double precision `float64` for numerical stability in mixture-model computations, hence $b_{\text{dtype}} = 8$. Table 8 reports the resulting estimates by decomposing the total memory into the four dominant terms above.

Table 8: Estimated peak memory footprint for fitting DP-GMM, decomposed into the dominant contributions.

Dataset	# Tokens (million)	D_{model}	K	Memory (GiB)				
				M_H	M_R	$M_{S^{(2)}}$	M_Σ	Total
WikiText-2	2.3	2,048	10	35.50	0.17	0.31	0.31	36.30
			10,000	35.50	173.35	312.50	312.50	833.85
WikiText-103 (Sub)	2.5	2,048	10	38.15	0.19	0.31	0.31	38.96
			10,000	38.15	186.26	312.50	312.50	849.41
COCO	7.4	768	10	42.33	0.55	0.04	0.04	42.97
			10,000	42.33	551.14	43.95	43.95	681.36
VQA V2	6.5	768	10	37.03	0.48	0.04	0.04	37.60
			10,000	37.03	482.10	43.95	43.95	607.02

D Baseline Selection

We select the original backbones (Grattafiori et al., 2024; Li et al., 2022) with their standard prediction heads as the main baselines for our experiments. This provides the most controlled comparison for our setting: our method keeps the backbone frozen and replaces the conventional black-box prediction head with an interpretable prototype-based prediction layer. Therefore, comparing against the original backbone isolates the effect of this replacement while holding fixed the pretraining data, architecture, representation space, and downstream evaluation protocol. The resulting performance difference directly measures the cost of introducing prototype-grounded interpretability.

We do not directly compare to kNN-LM (Khandelwal et al., 2020) or Sparse Neural Editor (SNE) (He et al., 2020), because these methods address different goals and operate under different assumptions. kNN-LM uses retrieval from a large external datastore to improve generation quality. In contrast, our work asks whether the black-box prediction head of a frozen language model can be replaced by a compact and scalable token-level prototype layer. In our method, predictions are produced directly from similarities to learned prototypes, so the prototypes are not post-hoc explanations or auxiliary retrieved evidence, but part of the prediction mechanism itself. Table 5 further shows that changing the matched prototypes substantially changes the output. As for SNE, it is closer in spirit. However, it retrieves and edits sentence-level prototypes, while our method performs free next-token generation using token-level prototypes whose similarities directly determine predictions. Thus, kNN-LM, SNE, and our method occupy different points in the design space: retrieval-augmented quality improvement, sentence-level prototype editing, and token-level prototype-grounded prediction, respectively.

This controlled backbone comparison is also consistent with the comparison strategy used in kNN-LM (Khandelwal et al., 2020) and SNE (He et al., 2020). kNN-LM evaluates its nearest-neighbour augmentation against the same underlying base LM, thereby isolating the effect of adding token-level retrieval. Similarly, SNE primarily compares against neural language model and neural-editor baselines within its own sentence-level prototype-editing setting. Following this principle, our comparison with the original head on the same frozen backbone is intended to isolate the cost of replacing the standard head with an interpretable prototype layer.