Scaling Inherently Interpretable Language Models

Guide Labs Team

Abstract

Interpretability is often treated as a tax on capability: language models are trained as opaque systems, then explained after the fact, with methods whose reliability is difficult to establish. We challenge this premise. Rather than reverse-engineering a trained model, we develop a recipe that makes interpretability a first-class constraint of the training pipeline, optimized alongside the language modeling objective. Across three orders of magnitude of compute, on both autoregressive and diffusion language models, this recipe scales with capability rather than against it. Surprisingly, model representations become more disentangled and aligned with human-understandable concepts with scale.

We instantiate the recipe with Steerling-8B, a diffusion language model with a causal attention mask. For any group of generated tokens, Steerling-8B attributes the output to relevant input tokens, human-understandable concepts, and training data. This enables closed-loop intervention: diagnose an output through its concept or feature attribution, retrieve similar training data, and correct the behavior through concept steering without retraining. Steerling-8B remains competitive with open peer models trained on substantially 2–16× more compute, suggesting a different scaling paradigm: interpretability can be designed into training, and it improves with scale.

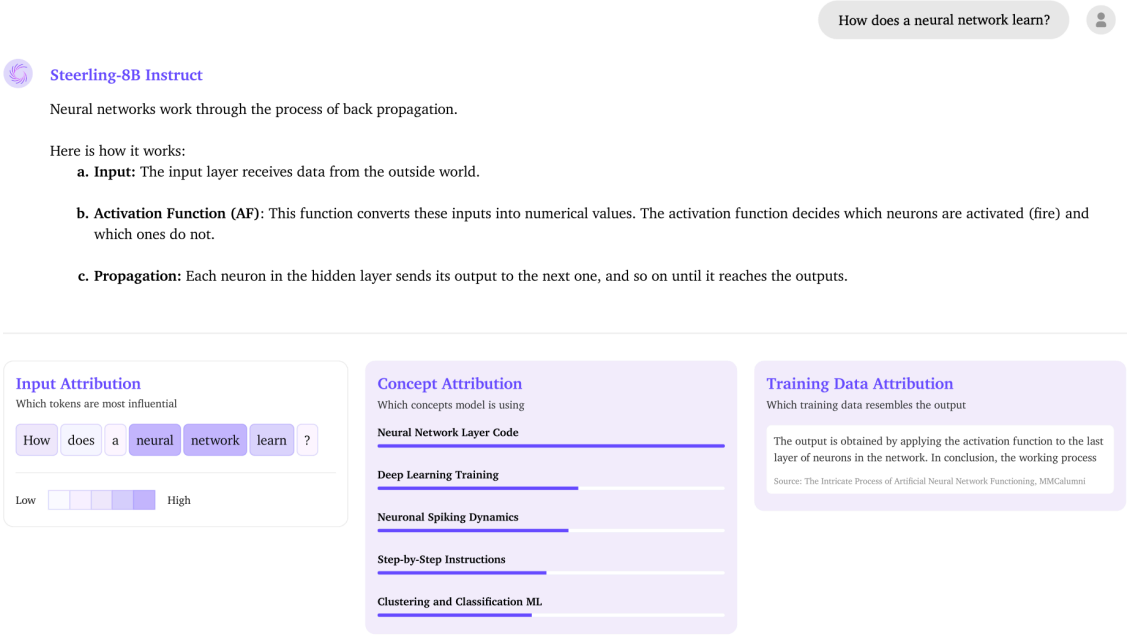

Figure 1. For the response shown, the model traces which prompt tokens mattered (*Input Attribution*), which internal concepts drove the output and by how much (*Concept Attribution*), and which training data the output resembles, with sources (*Training Data Attribution*).

Contents

1 Introduction

The most capable AI systems are also the least understood. This is often treated as the price of progress: if a model is constrained toward human-meaningful structure, the assumption goes, it must have weaker performance. The consequence is a now-pervasive workflow: we train the most capable model we can, then try to reverse-engineer it after the fact, as if training were a law of nature whose results we can only observe, never intervene on, lest we harm performance. We put this premise to the test. Across more than three orders of magnitude of compute, on both autoregressive and diffusion backbones, we show that building interpretability constraints into the training pipeline introduces a small, fixed scaling offset rather than a growing penalty. More surprisingly, we find that training with interpretability by design produces models whose representations become *more* disentangled and *more* aligned with human-understandable concepts as compute scales. **Our recipe makes models more interpretable as they become more capable.**

Reverse engineering models and post-hoc interpretability. The field's response is post-hoc interpretability: train a model, then inspect it with classifier probes (Alain and Bengio, 2016), feature attributions (Simonyan et al., 2013; Sundararajan et al., 2017b), sparse autoencoders (Bricken et al., 2023a), perturbation tests (Zeiler and Fergus, 2014; Lundberg and Lee, 2017), or chain-of-thought (Wei et al., 2022b; Nye et al., 2021). These tools can be useful, but they share a structural limitation: they explain a model that was never trained to make the explanation itself valid. A probe reports that information is decodable, not that the model uses it. Feature attribution often measures local sensitivity, not necessarily the effect of a human-relevant intervention. A sparse feature may reconstruct an activation without corresponding to a causal unit in the computation. A chain-of-thought may describe a plausible reason without being tied to the computation that produced the answer. The problem is that the standard model training recipe does not create an interface whose variables, interventions, and semantic labels are coupled to the prediction computation. Section 3.2 examines these failures in detail.

Inherent interpretability. Steerling-8B changes the training recipe. Instead of asking how to explain an opaque model after training, we ask what conditions must hold for an explanation to be faithful, then build those conditions into the data, architecture, objective, and loss functions. Section 3 formalizes this as *inherent interpretability*: an attribution is not an auxiliary visualization but a trained interface satisfying five conditions. It must be native to the forward computation; agree with the effects of valid interventions; use interventions supported by the training distribution; map attributed variables to human-understandable descriptions; and carry enough predictive load.

Three attribution axes. Steerling-8B instantiates this recipe as a diffusion language model. For any output, the model traces its prediction along three axes:

1. to input features that affect the output under a trained absence baseline;
2. to human-understandable concepts in its internal representations that contribute directly to the output; and
3. to training data.

From explanation to control. The same interfaces support intervention. Each concept embedding is a direction the model already uses in its forward pass, so amplifying or suppressing a concept is a closed-form edit. This makes systematic a closed loop that standard post-hoc workflows do not provide by construction: decompose an output into concept contributions, surface training data the

model represents as similar, edit the relevant concept, and verify the resulting change, all without retraining.

Atlas. A central obstacle to this recipe is that no suitable concept library existed at the scale of modern pretraining corpora. We build Atlas, a large-scale concept annotation pipeline that starts from millions of documents, extracts hundreds of millions of free-form tags, canonicalizes them into over 33,000 concepts, and trains an annotator that labels arbitrary text at chunk level. In total, Atlas annotates over 1 Trillion tokens across web text, code, mathematics, and academic prose. Section 4 describes Atlas.

Architecture. The architecture makes these concepts native to the model's computation. Steerling-8B uses a backbone with block-causal attention that preserves diffusion-style parallelism within blocks while retaining autoregressive-style KV caching across blocks. Between the transformer backbone and the language-modeling head, we insert an additive concept bottleneck that makes the logit decomposition algebraically exact. The masking objective gives the model a trained representation of "no information at this position," making feature-removal baselines in-distribution by construction. The overhead is small and decreases with scale: the concept module accounts for 4% of parameters at 8B, and under the same parameterization, would fall below 1% at frontier scales. Section 5 describes the architecture and training objective.

Scaling. The main empirical question is whether this structure makes the model weaker. We answer with IsoFLOP scaling sweeps across three orders of magnitude of compute and four model families: autoregressive, causal diffusion, autoregressive with concepts, and causal diffusion with concepts. Adding the concept module shifts the compute-optimal scaling exponents by a small, fixed per-backbone offset; the cost of interpretability does not grow with scale. Simultaneously, all interpretability metrics improve with compute on both backbones: the model predicts concepts more accurately, separates known and unknown representations more cleanly, routes more of its prediction through concepts rather than the residual, and aligns its concept embeddings more closely with human-meaningful labels. The validation loss of Steerling-8B is predicted within 0.11 nats from small-scale fits using the joint Chinchilla form, and three of four interpretability metrics are predicted within tight bounds. Under the metrics we measure, the model does not become harder to understand as it scales. It becomes easier. Section 8 presents the full analysis.

An interpretable model can be competitive with opaque peers trained on far more compute. We train Steerling-8B on 1.2 trillion tokens followed by 150 billion midtraining tokens on a code and math augmented mixture. Compared with open peer models at similar parameter scale, each trained on roughly 2–16$\times$ more compute, Steerling-8B lands within approximately 10% of their average benchmark performance, despite carrying interpretability constraints throughout training. A model can be both interpretable and competitive. Sections 9 and 10 describe the full training process.

Overview. The remainder of the paper is organized as follows. Section 2 reviews transformers, diffusion language models, and concept bottleneck models. Section 3 presents the interpretable training recipe and formalizes faithfulness and inherent interpretability. Section 4 describes Atlas, the concept annotation pipeline. Section 5 describes the causal-diffusion architecture and concept module. Section 6 presents attribution and steering capabilities. Section 7 defines the interpretability metrics. Section 8 presents the scaling-law analysis. Sections 9 and 10 describe Steerling-8B pretraining and midtraining. Section 11 discusses related work, and Section 12 concludes.

2 Background

In this section, we present material that is core to our discussion in the rest of the paper. In addition, we present the notation that we use across the rest of the work.

2.1 Transformer notation

A Transformer (Vaswani et al., 2017) maps a sequence of input tokens to a sequence of hidden states per position. Each hidden state summarizes the token at that position together with its contextual information, and the model projects these hidden states into logits over the vocabulary. We summarize the notation in Table 1.

Symbol	Meaning
$\mathbf{x} = (x^1, \ldots, x^N)$	Input token sequence of length N
x^i	Token at position i
V	Vocabulary
$T_{x^i} \in \mathbb{R}^d$	Learned embedding of token x^i
d	Hidden dimension
L	Number of transformer layers
$h \in \mathbb{R}^d$	Transformer hidden state at a given position
$W \in \mathbb{R}^{\lvert V\rvert \times d}$	Language modeling head
W_y	Row of W corresponding to token y
$\ell_y = h^\top W_y$	Logit for output token y
p_θ	Model with parameters θ
$\mathbf{x}^{<i}$	Sub-sequence of tokens before position i
$\mathcal{L}_{\text{AR}}$	Autoregressive training loss
$\mathbf{x}_t$	Corrupted sequence at noise level t
$t \in [0, 1]$	Noise level
$M(\mathbf{x}_t)$	Set of masked positions in $\mathbf{x}_t$
$\mathcal{L}_{\text{MDM}}$	Masked diffusion training loss

Table 1. Transformer and language-model notation.

2.2 Autoregressive language models

Autoregressive language models (Radford et al., 2018) are trained on the task of next-token prediction: given an input sequence, the model predicts the next token, one at a time, conditioned on all previous tokens. We use the following notation:

- $p_\theta(x^i \mid \mathbf{x}^{<i})$ is the model's predicted distribution over the next token at position i, given all previous tokens $\mathbf{x}^{<i} = (x^1, \ldots, x^{i-1})$, where θ denotes the model parameters.
- The training loss is the negative log-likelihood of the next token, averaged over all positions in the sequence:

$$\mathcal{L}_{\text{AR}} = -\frac{1}{N} \sum_{i=1}^{N} \log p_\theta(x^i \mid \mathbf{x}^{<i}). \tag{1}$$

2.3 Masked diffusion language models

Diffusion language models (Austin et al., 2021a; Ou et al., 2024; Sahoo et al., 2024; Shi et al., 2024) are trained by reversing a forward corruption process applied to the input sequence, rather than by predicting the next token. A common variant is the masked diffusion language model (MDM), where the corruption process independently replaces tokens with a special `[MASK]` token. The model is trained to reconstruct the original tokens from the corrupted sequence. The amount of corruption is controlled by a noise level $t \in [0, 1]$, where each token is masked with probability t. Thus, $t = 0$ corresponds to no corruption, while $t = 1$ corresponds to the fully masked sequence. We use the following notation:

- $\mathbf{x}_t$ is the corrupted sequence at noise level t.
- $M(\mathbf{x}_t)$ is the set of positions in $\mathbf{x}_t$ that have been masked.
- $p_\theta(x^i \mid \mathbf{x}_t)$ is the model's predicted distribution over the original token at masked position i, given the corrupted sequence.

The training objective is a cross-entropy loss over the masked positions only, averaged over noise levels and sequences:

$$\mathcal{L}_{\text{MDM}} = \mathbb{E}_{t,\mathbf{x},\mathbf{x}_t}\left[\frac{1}{|M(\mathbf{x}_t)|}\sum_{i\in M(\mathbf{x}_t)} -\log p_\theta(x^i \mid \mathbf{x}_t)\right]. \tag{2}$$

2.4 Concept bottleneck models

Concept Bottleneck Models (CBMs) (Koh et al., 2020) add interpretability to a black-box neural network by inserting a layer of human-interpretable concepts between the input and the output. A sample x is first mapped to concept activations $c = \phi(x) \in \mathbb{R}^n$, where each entry of c corresponds to a supervised, human-interpretable concept. A second function ψ then predicts the label from the concepts, $y = \psi(c)$. When ψ is linear, as in the original formulation, the final prediction is a weighted sum of interpretable concepts:

$$x \xrightarrow{\phi} c \xrightarrow{\psi} y. \tag{3}$$

Training a CBM requires ground-truth concept labels c alongside the target y, giving two losses. A concept loss matches $\phi(x)$ to c, and a prediction loss matches $\psi(\phi(x))$ to y.

Concept Bottleneck Generative Models (CBGMs) (Ismail et al., 2024) extend this idea to generative modeling. Since a fixed set of supervised concepts cannot capture everything in the input, CBGMs add an unsupervised concept channel u alongside the known concepts c, so that generation routes through both an interpretable known part and an unknown part that absorbs the remaining information:

$$x \xrightarrow{\phi} (c, u) \xrightarrow{\psi} y. \tag{4}$$

CBGMs introduce an additional orthogonality loss, which encourages the unknown embeddings to be orthogonal to the known concept embeddings so that the unknown channel encodes information distinct from the supervised concepts.

3 A recipe for building interpretable models

In this section, we present a general recipe for building interpretable language models. The recipe follows the standard pipeline for training language models (data curation, architecture and loss design, optimization, and evaluation) but modifies each stage to introduce human-interpretability constraints.

Roadmap. We proceed in four parts:

- Section 3.1 defines the interpretability requirements that we aim to satisfy in this work.
- Section 3.2 shows why the standard recipe does not provide the required guarantees.
- Section 3.3 formalizes explanation faithfulness and inherent interpretability.
- Section 3.4 presents the interpretable recipe: pipeline modifications traced to specific conditions (summarized in Figure 2).

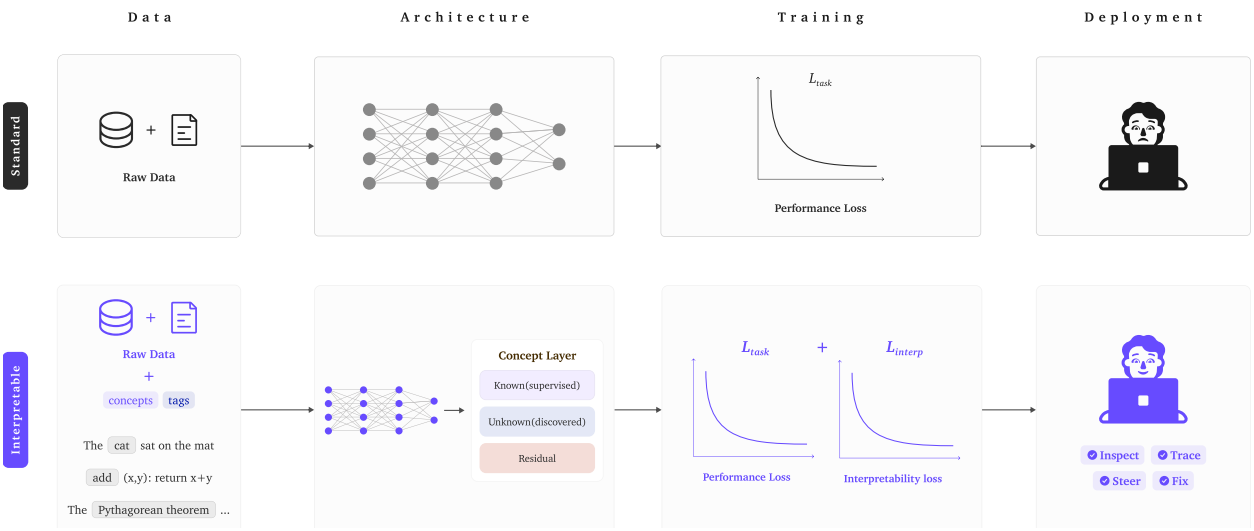

Figure 2. The recipe for training an interpretable model.

3.1 Interpretability requirements

Here we formalize our interpretability requirements: *input attribution*, which measures the effect of removing each input token; *concept attribution*, which decomposes the model's output into per-concept contributions; and *training data (similarity) attribution*, which retrieves training examples the model considers similar to its output.

Interpretability describes the interface between a model and a human: the structures through which a person can inspect, evaluate, understand, and act on a model's behavior. We require three specific capabilities of this interface.

Informal Interpretability Requirements

Given a model, p_θ, for any output it produces, we want a human user to be able to answer three questions:

1. **"What in the input mattered?"** This requirement asks which parts of the input most influence the output?
2. **"What topics/ideas are responsible for that?"** This requirement asks what topics/ideas in the model's internal representation drove this output?
3. **"Where is the output coming from?"** This requirement asks which training data resembles what the model just produced.

The literature refers to such artifacts as explanations, interpretations, or attributions. In this work, we use *attribution* throughout: each requirement asks the model to attribute to a specific component.

We translate each question into a precise operation, which we term an *attribution*:

1. **"What in my input mattered?"** $\rightarrow$ **Input attribution**. We can measure the change in the model's output when each input is replaced by a trained absence baseline, and rank tokens by the magnitude of this change.
2. **"What topics/ideas are responsible for that?"** $\rightarrow$ **Concept attribution**. We can decompose the model's internal representation into contributions from human-understandable topics, and trace each topic's contribution to the output.
3. **"Where is this coming from?"** $\rightarrow$ **Training data attribution**. We can retrieve the training examples whose representations are most similar to the representation of the model's output.

To move forward, we will now concretize core definitions as we specify the components necessary to build an inherently interpretable language model.

Concepts. In this work, and as is common in the literature, we term a human-understandable topic a *concept*. More generally, a concept is a coherent unit of meaning that a human recognizes and that we seek to associate with the model's internal representations. Concepts span multiple levels of granularity: high-level themes like `sports`, `politics`, or `tourism`; mid-level topics like `gradient descent optimization`, `state elections`, or `Mediterranean cuisine`; and fine-grained units like `dropout regularization in recurrent networks` or `ranked-choice ballot counting`. They may also capture stylistic or functional attributes, `formal tone`, `sarcasm`, `step-by-step reasoning`, rather than topical content. The defining property is not the level of abstraction but human recognizability: a domain expert, given sufficient context, can identify whether the concept is present in a piece of text. Definition 3.1 formalizes this intuition.

Definition 3.1 (Concept)**.** A *concept* is a pair $c = (z_c, m_c)$ consisting of two components:

- z_c, **what the model computes with**: an activation, embedding, or dimension that participates in producing the model's output, ℓ_y, and can be modified to change the model's behavior (formalized in Section 3.3.2).
- m_c, **what the human sees**: a description that explains, in natural language, what z_c represents.

In this work, $m_c = (l_c, d_c, \mathcal{T}_c)$, where l_c is a short label, d_c is a one-sentence description, and $\mathcal{T}_c$ is a set of characteristic words grounding the concept in observable language.

We refer to z_c as the concept's *model variable* and m_c as its *semantic card* throughout the paper. The data pipeline (Section 4) produces semantic cards before any model exists; the model variable z_c is instantiated when the architecture described in Section 5 is trained.

With the definition of a concept in place, we now make each of our three attributions—input, concept, and training data—mathematically precise. We do this for a specific reason: in Section 3.2, we will show that post-hoc methods fail to deliver these attributions reliably, and in Section 3.3, we will define formal conditions under which they *are* reliable.

Notation. Let p_θ be the model with parameters θ, $\mathbf{x} = (x^1, \ldots, x^N)$ an input sequence of N tokens, y a predicted output token, and $\ell_y = h^\top W_y$ the output logit for token y, where $h \in \mathbb{R}^d$ is the hidden state and W_y is the row of the language modeling head corresponding to y. We write $\mathcal{A}$ for a generic attribution functional, and use superscripts to distinguish the three types.

Input attribution. *If an input token were absent, how much would the output change?* The input attribution functional maps the model, input, and output to a vector of per-token relevance scores:

$$\mathcal{A}^{\text{input}}(p_\theta, \mathbf{x}, y) = \big(\alpha_1, \ldots, \alpha_N\big) \in \mathbb{R}^N,$$

where

$$\alpha_i = \ell_y(\mathbf{x}) - \ell_y(\mathbf{x}_{\setminus i})$$

measures the effect of replacing token, x^i, with a baseline representing absence. This definition has a prerequisite that we formalize later: it is meaningful only if $\mathbf{x}_{\setminus i}$ is in-distribution. If the model has never seen inputs with position i absent, the output $\ell_y(\mathbf{x}_{\setminus i})$ reflects extrapolation. We return to this in Section 3.3.1, where we define *validity*.

Concept attribution. *Which concepts contributed to the model's prediction, and by how much?* The concept attribution functional maps the model, input, and output to a vector of per-concept relevance scores:

$$\mathcal{A}^{\text{concept}}(p_\theta, \mathbf{x}, y) = \big(\phi_1(y), \ldots, \phi_n(y), \rho(y)\big) \in \mathbb{R}^{n+1},$$

where $\phi_c(y)$ is the contribution of concept c to the output logit ℓ_y and $\rho(y)$ is the residual; the portion of the prediction not attributed to any concept. The output logit decomposes as:

$$\ell_y = \sum_c \phi_c(y) + \rho(y).$$

How the decomposition above is achieved, is an architectural choice described in Section 5.

Training data (similarity) attribution. *Which training examples does the model consider similar to this output?* The similarity attribution functional maps the model output and input to a set of training data, ranked by similarity, under the model's own encoder r_θ:

$$\mathcal{A}^{\text{retrieval}}(p_\theta, \mathbf{x}) = \text{TopK}_{D \in \mathcal{D}} \ \text{sim}\big(r_\theta(\mathbf{x}), r_\theta(D)\big).$$

Similarity not causation. Our training data attribution requirement does not seek causation. It only requires that the model represents these training data similarly to its output. We do not claim training influence, which would require approximating counterfactuals over retraining (Koh and Liang, 2017; Bae et al., 2022; Grosse et al., 2023).

Retrieval differs from the other two attributions in an important way: it is a *representational provenance* functional, not a causal attribution. Concept attribution decomposes the output; input attribution measures counterfactual effects; retrieval reports similarity in the model's own geometry. Its criterion for being meaningful is that the similarity scores reflect the model's actual representation, not an externally imposed metric. This holds by construction when the retrieval encoder is the same encoder that feeds into prediction.

3.2 Why post-hoc methods fail to satisfy the interpretability requirements

Each of the three attributions defined above has post-hoc analogues: methods that attempt to compute input, concept, or training data attributions from an already-trained model, without modifying its training in any way. Each can in principle succeed. Each, in standard practice, does not *guarantee* the required conditions because the model was not trained to support them.

We begin with the root cause: standard training does not create an explanatory interface whose variables are coupled to prediction. We then examine post-hoc attempts at each attribution in turn: concept attribution via probes and sparse autoencoders, input attribution via gradient and perturbation methods, and training data attribution via influence functions. We close with the unifying diagnosis and a deeper obstacle, the Rashomon problem, which shows that even faithful post-hoc explanations can be arbitrary.

Root cause. The standard model training pipeline optimizes for prediction.

The core issue with post-hoc interpretability

A standard model is trained for prediction. Its training objective does not ask for representations that decompose into meaningful units, baselines that make "what if this input were absent?" a well-defined question, or reasoning traces that reflect the actual computation. Post-hoc interpretability methods assume one or more of these structures exist and attempt to recover them after training. But they were never created: the model's representations have no reason to be interpretable, its responses to missing inputs have no reason to be meaningful, and its verbal explanations have no reason to match its internal computations. The failures below are not limitations of specific methods; they are consequences of analyzing a model that was built without specifying formal interpretable conditions.

Probes. A common approach to post-hoc concept attribution is to train a linear classifier on the hidden states h of an already-trained model and interpret high accuracy as evidence that the model "represents" a concept (Alain and Bengio, 2016; Belinkov, 2022). The fundamental gap is between information *presence* and information *use*: a probe asks whether concept c is decodable from h, not whether p_θ relies on c for prediction. A perfectly accurate probe is compatible with the model ignoring the probed feature entirely (Ravichander et al., 2021). Several lines of evidence sharpen this concern:

- Linear and structural probes recover information with high accuracy from randomized contextualized embeddings (Conneau et al., 2018; Hewitt and Manning, 2019).
- Syntactic probes do not generalize across domains (Maudslay and Cotterell, 2021).

- Probes extract features merely encoded in token embeddings but unused during inference (Ravichander et al., 2021).
- Capacity-controlled, information-theoretic, and amnesic probing mitigate some false discoveries but do not close the presence-versus-use gap (Voita and Titov, 2020; Elazar et al., 2021).

Even when probes identify directions useful for steering, one cannot assert that the model uses those directions for prediction. Each new concept requires training a separate probe, with no guarantee of coherence across probes.

Sparse autoencoders. Sparse autoencoders (SAEs) decompose a frozen model's hidden states h into sparse dictionaries of features (Bricken et al., 2023b), and have surfaced striking results: features corresponding to the Golden Gate Bridge, emotional states, and safety-relevant behaviors (Templeton, 2024). However, the gap between discovering interesting features and guaranteeing that those features are the variables through which p_θ computes its predictions is substantial. Because the SAE is trained on a frozen model, its features are post-hoc descriptions with no structural relationship to the prediction pathway. More concerning, recent work shows that SAEs trained with different random seeds produce different feature sets, and that random SAE baselines match fully-trained SAEs on sparse probing and causal editing metrics (Korznikov et al., 2026). The explanations may seem convincing, but they are not uniquely grounded in what the model learned.

Gradient-based input attribution. Gradient-based attribution methods compute $\partial \ell_y / \partial x^i$ and present the result as a measure of feature importance. Several variants abound including saliency maps (Simonyan et al., 2013), Grad-CAM (Selvaraju et al., 2016), and integrated gradients (Sundararajan et al., 2017b), and many others.

Gradients measure infinitesimal sensitivity; however, the human typically cares about the effect of removing or substantially changing a feature, a finite but larger perturbation. For gradients to be a useful proxy, the model must be approximately linear over the perturbation budget the human cares about or be off-manifold robust (Dombrowski et al., 2019; Srinivas et al., 2022, 2023). However, unless the training procedure is regularized to produce models that are locally linear, there is no incentive for the model to satisfy this requirement. Handed a model, the user might not have a way to assess how locally linear it is, and therefore no way to know whether the gradient is a valid proxy for the perturbation they care about.

Perturbation-based input attribution. Perturbation-based methods like SHAP (Lundberg and Lee, 2017) and occlusion (Zeiler and Fergus, 2014) replace input tokens with a reference value and measure the output change. Common reference values (zero vectors, random tokens, arbitrary masks) are out of distribution: the observed effect reflects the model's extrapolation behavior, not the feature's actual role (Hooker et al., 2019; Kumar et al., 2020; Jain et al., 2022). The resolution is to choose a null state that is in-distribution by construction for the model at hand.

Unfaithful chain of thought. Chain-of-thought reasoning (Nye et al., 2021; Wei et al., 2022b) produces natural-language explanations alongside outputs. The model's stated reasoning need not reflect its actual computation: a model can produce a correct-sounding chain that arrives at the answer for entirely different internal reasons, or a plausible chain that is confabulated (Lanham et al., 2023; Madsen et al., 2024a; Turpin et al., 2023; Korbak et al., 2025; Chen et al., 2025). The explanation channel (generated text) and the computation channel (hidden states $\rightarrow$ output) are not structurally coupled by training.

A deeper problem: explanatory multiplicity. Even if post-hoc methods could deliver faithful attributions for a given model, the result would be contingent on which model happened to emerge from training. Models with identical performance can differ arbitrarily in their internal mechanisms, and explanatory multiplicity is decoupled from predictive multiplicity: two models with indistinguishable accuracy can produce contradicting attributions for the same input (D'Amour et al., 2022; Brunet et al., 2022). This instability propagates to any downstream action, e.g. steering, editing, counterfactual recommendations, derived from the explanation (Pawelczyk et al., 2020). The recipe in Section 3.4 addresses this by anchoring the explanatory interface: the concept library, the absence baseline, and the additive decomposition are shared across all models the training procedure can produce. Different runs yield different parameters, but the interface through which attributions are computed is fixed. Section 11 discusses the underspecification and Rashomon literature in detail.

The common thread

Every gap above traces to the same structural absence: the model was not trained to make the attribution valid. The training was indifferent to whether probed features would be causally relevant, perturbation baselines would be in-distribution, gradients would be informative, or stated reasoning would reflect computation. This observation is not new (Méloux et al., 2025): post-hoc attributions are not identifiable estimators for the quantities they claim to measure. What is needed is to make the attributions identifiable, and one way to do that is to build the attributions into the training process itself.

3.3 Faithfulness and inherent interpretability

The gaps identified in Section 3.2 share a common structure: the attribution's claims either disagree with what happens when one intervenes on the model, or the intervention itself is invalid. We now formalize these two failure modes as conditions on attributions (Section 3.3.1), then define the class of models that satisfy them by construction (Section 3.3.2).

3.3.1 Faithfulness

Prior work on faithfulness. The concept of attribution faithfulness has been widely discussed in the literature (Hooker et al., 2019; DeYoung et al., 2020; Lyu et al., 2024; Atanasova et al., 2023). Most treatments define faithfulness informally as "the explanation/attribution reflects the model's actual reasoning" (Jacovi and Goldberg, 2020) or operationalize it through specific tests, perturbation checks, sufficiency, comprehensiveness (DeYoung et al., 2020), without unifying these into conditions that a training procedure can be designed to satisfy. We distill the literature into two concrete conditions, agreement and validity, that are checkable, traceable to specific design choices, and help to diagnose every post-hoc failure identified in Section 3.2.

Definition 3.2 (Faithfulness)**.** An attribution functional, $\mathcal{A}$, is *faithful* with respect to model, p_θ, intervention family, $\mathcal{I}$, and tolerance ξ, if:

1. **Agreement.** The output relevance scores, from the attribution, predict the observed effects of interventions in $\mathcal{I}$ within tolerance ξ.
2. **Validity.** The interventions in $\mathcal{I}$ belong to a *training-supported intervention family*: the model has been trained on, or explicitly regularized for, the perturbed states used to define the attribution.

Two types of interventions. We distinguish input interventions $\mathcal{I}_x$, such as replacing token, x^i, with the mask token, from latent interventions $\mathcal{I}_z$, such as modifying a concept variable z_c. For masking,

validity means the model saw masked contexts during training. For concept interventions, validity means edits are bounded to observed activation ranges or sampled concept values.

Mapping to Section 3.2. Each post-hoc issue, previously identified, is a violation of one or both conditions. Probes violate agreement: information presence does not imply information use. SAEs violate agreement: reconstruction geometry does not imply causal role. Gradient methods violate agreement: local sensitivity does not match the human's perturbation budget. Perturbation methods violate validity: the baseline is out of distribution. Chain-of-thought violates agreement: the text channel is not coupled to the computation channel.

3.3.2 Inherent interpretability

We now have all the relevant definitions and terms to state what we mean by inherent interpretability in this work.

Definition 3.3 (Inherent interpretability)**.** A model, p_θ, is *inherently interpretable* with respect to an attribution functional, $\mathcal{A}$, an intervention family, $\mathcal{I}$, a semantic map $\mathcal{M} : c \mapsto m_c$ assigning each concept its semantic card (Definition 3.1), a tolerance ξ, and a coverage threshold ζ if its training (or finetuning) procedure produces models for which:

Causal faithfulness:

1. **Nativeness.** The attributed variables are part of the computation that produces the model's output, ℓ_y.
2. **Agreement.** The attribution predicts the effect of an intervention before it is applied. For an intervention in $\mathcal{I}$, let $\Delta\ell_y$ be the *observed* change: the difference in the output logit between a forward pass with the intervention applied (e.g., a token replaced by mask, or a concept variable modified) and the unmodified forward pass. Let $\hat{\Delta}\ell_y$ be the *predicted* change: the change implied by the attribution's relevance scores alone, computed without running the intervened forward pass. Agreement requires
$$|\Delta\ell_y - \hat{\Delta}\ell_y| \leq \xi.$$
3. **Validity.** Interventions in $\mathcal{I}$ belong to a training-supported intervention family.

Semantic faithfulness:

4. **Interpretation.** The semantic map $\mathcal{M}$ gives human-valid descriptions of the attributed variables: each card m_c accurately describes what its variable z_c encodes.
5. **Coverage.** The attributed variables together account for at least a fraction ζ of the model's prediction, with the unattributed residual $\rho(y)$ explicitly reported.

Inherent Interpretability does not mean total mechanistic transparency. Inherent interpretability is a targeted guarantee for specified attribution queries, not a global claim about every neuron or attention head. When we say inherent interpretability in this work, we do not claim that the entire model is human-understandable.

Causal vs. semantic faithfulness

Conditions 1–3 establish *causal faithfulness*: the attribution correctly describes the model-side variable z_c and its role in producing ℓ_y. Conditions 4–5 establish *semantic faithfulness*: the human description m_c validly represents z_c, and the interpretable component is useful for the output.

Concept leakage (Mahinpei et al., 2021) is a failure of semantic faithfulness: the attribution may correctly predict the causal effect of z_c (conditions 1–3 hold) while m_c incompletely describes what z_c encodes (condition 4 fails). For example, if z_c encodes "sports" plus hidden information about "gender," the numeric attribution $\phi_c(y)$ correctly predicts the effect of modifying z_c, but the semantic card $m_c =$ "sports" is incomplete.

Examples of inherent interpretability. The definition is agnostic to the form of explanation. We illustrate with examples in the literature:

1. **Masking → inherently interpretable w.r.t. $\mathcal{A}^{\text{input}}$.** Training with a masking objective places the absence baseline in-distribution, satisfying validity. Madsen et al. (2024b) make this move for finetuning; we extend it to pretraining.

2. **Concept bottleneck → inherently interpretable w.r.t. $\mathcal{A}^{\text{concept}}$.** An additive bottleneck gives nativeness and exact agreement ($\xi = 0$). Koh et al. (2020) introduced this for classification. And Ismail et al. (2024) extend that approach to generative models.

3.4 The interpretable model-training recipe

We now have three attribution functionals (Section 3.1), a demonstration that post-hoc methods do not guarantee them (Section 3.2), and a formal definition of inherent interpretability specifying what is needed (Definition 3.3). The recipe modifies the standard training pipeline at exactly the points needed to satisfy the five conditions. Every modification traces to a specific condition; Figure 3 summarizes the full pipeline.

Why pretraining. Interpretability constraints are most effective when present during representation formation. Finetuning restructures existing representations; pretraining shapes them from the start. Our recipe targets pretraining, though the contracts apply to finetuning with weaker guarantees.

The standard recipe. A standard training pipeline trains a model p_θ on input-output pairs $\{(\mathbf{x}_i, y_i)\}$ by minimizing a task loss, and evaluates on downstream metrics. This recipe is effective for prediction but does not require the model to expose explanations.

The interpretable recipe. The interpretable recipe modifies the standard pipeline at five points (right column of Figure 3 and bottom row of Figure 2). Each modification is motivated by a specific condition of Definition 3.3.

Concept annotations (Interpretation). The training data is augmented with concept annotations: $\mathcal{D} = \{(\mathbf{x}_i, y_i, \mathbf{c}_i)\}$. Without supervised targets, the model has no signal to organize its representations around human-meaningful categories; it will default to whatever geometry minimizes the task loss (Section 3.2). Concept annotations provide the semantic targets that ground the interpretation condition: each concept variable, z_c, has a corresponding semantic card, m_c, whose meaning is established before training. The annotations also enable a control interface, since concepts with prior semantics can be monitored, amplified, or suppressed. *Our instantiation:* Section 4 describes the Atlas pipeline that produces concept annotations.

	Standard Training Recipe	Interpretable Training Recipe
Data	$\mathcal{D} = \left\{ \overbrace{(\mathbf{x}_i, y_i)}^{\text{Input–Output}} \right\}_{i=1}^{n}$	$\mathcal{D} = \left\{ \overbrace{(\mathbf{x}_i, y_i)}^{\text{Input–Output}}, \overbrace{\mathbf{c}_i}^{\text{Explanations}} \right\}_{i=1}^{n}$
Arch.	$\ell_y = \underbrace{h}_{\text{Opaque}}{}^{\top} W_y$	$\ell_y = \underbrace{\sum_c \phi_c(y)}_{\text{Bottleneck}} + \underbrace{\rho(y)}_{\text{Residual}}$ → concept & retrieval attribution
Objective	$\mathcal{L}_{\text{AR}}$ (§2.2)	Objective with **trained absence baseline** → input attribution
Loss	$\sum_i \ell\big((\mathbf{x}_i, y_i); \theta\big)$	$\underbrace{\sum_i \ell_{\text{task}}}_{\text{Task}} + \underbrace{\sum_j \lambda_j \ell_{\text{interp},j}(\mathbf{c}_i)}_{\text{Interpretability}}$
Evaluate	$\underbrace{\text{score}}_{\text{Task Metric}}$	$\underbrace{\text{score}_{\text{task}}}_{\text{Task}}$, $\underbrace{\text{score}_{\text{interp}}}_{\text{Interpretability}}$
Attribute	$\underbrace{\hat{\mathcal{A}}^{\text{post-hoc}}(p_\theta, \mathbf{x}, y)}_{\text{Approximate (§3.2)}}$	$\underbrace{\mathcal{A}^{\text{input}}}_{\text{In-dist.}}$, $\underbrace{\mathcal{A}^{\text{concept}}}_{\text{Exact}}$, $\underbrace{\mathcal{A}^{\text{retrieval}}}_{\text{Native}}$

Figure 3. The standard and interpretable recipes, compared stage by stage (Section 3.4). The interpretable recipe augments each stage to satisfy the conditions of Definition 3.3. Each attribution functional (bottom row, right) traces to a specific upstream modification (green arrows): the bottleneck architecture enables exact concept attribution and native retrieval in the model's own concept-aligned geometry; the trained absence baseline in the objective makes feature attribution in-distribution. The standard recipe supports only post-hoc attribution (orange), which does not guarantee the required conditions (Section 3.2).

Bottleneck architecture (Nativeness, Agreement). A model architecture that routes the model's predictions through concepts makes it so that each concept's contribution to the output can be directly computed from the forward pass. This is a structural requirement: the concept variables, z_c, must lie on the computational path from the model's internal state to the output. The specific mechanism, additive bottleneck, multiplicative gating, or another decomposition, is an implementation choice. Section 5 describes an additive concept bottleneck with a linear output head.

Trained absence baseline (Validity). Input attribution requires replacing input features with a baseline representing absence and measuring the change in output. For this to be meaningful, the model must have a learned representation of "no information at this position". Without this, the perturbed input $\mathbf{x}_{\setminus i}$ is out of distribution, and the observed output change reflects extrapolation rather than the feature's actual role. The contract is that the training objective must include the perturbed states used to define $\mathcal{A}^{\text{input}}$. Standard autoregressive next-token objective does not typically incorporate this type of masking. *Our instantiation:* Section 5 describes a masked diffusion objective ($\mathcal{L}_{\text{MDM}}$) that trains the model on corrupted sequences, making the mask token an in-distribution absence baseline.

Interpretability Losses (Interpretation, Coverage). The architecture provides the slot for concepts; the losses ensure the slot is used. Two failure modes must be prevented. First, *concept leakage*: the concept variable z_c may encode information beyond what the semantic card m_c describes, degrading the interpretation condition. Concept losses that align z_c activations with the annotations

$\mathbf{c}_i$ counteract this by maintaining semantic alignment between the model-side variable and its human description. Second, *residual domination*: the residual $\rho(y)$ may absorb most of the predictive capacity, leaving the concept decomposition algebraically exact but practically vacuous. Losses that penalize the residual, enforce reconstruction of the unexplained hidden state, and encourage independence between concept components address this. The general form is $\mathcal{L} = \mathcal{L}_{\text{task}} + \sum_j \lambda_j \mathcal{L}_{\text{interp},j}$, where the weights λ_j may be annealed during training to balance task performance and interpretability. *Our instantiation:* Section 5 describes the specific loss components and annealing schedule.

Evaluation. Task metrics (perplexity, accuracy) remain necessary to ensure interpretability is not obtained by discarding predictive information. In addition, the model should be evaluated against the five conditions of Definition 3.3:

1. **Causal faithfulness (Nativeness).** Are the attributed variables on the computational path? Agreement: do interventions on z_c change ℓ_y as predicted by $\phi_c(y)$? Validity: are the interventions in-distribution?
2. **Semantic faithfulness.** Interpretation: do humans independently recognize the concepts from the model's characteristic evidence? Coverage: what fraction of prediction mass is carried by concepts versus the residual?

Our instantiation: Section 4.4 describes the human recoverability study (Condition 4) and the quantitative evaluation (Conditions 2, 3, 5).

Summary. Every modification traces to a condition of Definition 3.3; Table 2 summarizes the consequence of removing each one. The recipe itself is not new: its elements are implicit in work on concept bottlenecks (Koh et al., 2020), interpretable-by-design architectures (Rudin, 2019), masking-based attribution (Madsen et al., 2024b), and recent theoretical treatments that derive interpretability constraints from explicit premises (Barbiero et al., 2026). Our contribution is to make the recipe explicit, trace each element to a formal condition, and instantiate it for language models at pretraining scale.

Remove…	Condition broken	Consequence
Concept annotations	Interpretation (4)	No semantic targets
Bottleneck architecture	Nativeness (1), Agreement (2)	$\phi_c(y)$ not computable
Trained absence baseline	Validity (3)	Input-attribution baselines OOD
Interpretability losses	Coverage (5), Interpretation (4)	Residual absorbs capacity

Table 2. Removing any single recipe modification breaks a specific condition of Definition 3.3.

This completes the recipe. The interpretability requirements (Section 3.1) defined what we want; the post-hoc analysis (Section 3.2) showed why the standard pipeline cannot deliver it; the definitions of faithfulness and inherent interpretability (Section 3.3) formalized what is needed; and the recipe above specified how to modify each stage of the pipeline to satisfy those conditions. The remainder of the paper instantiates this recipe: Section 4 instantiates the data contract, building the concept library; Section 5 instantiates the architecture, objective, loss, and optimization contracts, building the model; and Section 8 shows that these contracts preserve compute-optimal scaling while the interpretability properties themselves improve with compute. Finally, Sections 9 and 10 carry the recipe to full scale, pretraining and mid-training Steerling-8B.

4 Data

In this section, we describe the process of building large-scale concept-annotated pretraining, mid-training, and post-training corpora. As discussed in Section 3, to satisfy the concept interpretability constraints, we need supervision: data annotated with concepts at a level of granularity to capture meaning. Towards this end, we present **Atlas**, an automated system for annotating language modeling corpora with human-understandable concepts at a fine-grained level. Using Atlas, we annotated a 1.5 trillion-token corpus spanning web text, scientific writing, code, and synthetic data with over 33,000 concepts across science, technology, philosophy, medicine, law, etc.

Overview. First, we discuss why existing concept libraries are inadequate and state the desiderata that a suitable library must satisfy (Section 4.1). We then describe the Atlas pipeline: a three-stage process that moves from sampled documents to high-recall tags—short free-form words or phrases associated with local spans of text—(Section 4.2.1), from tags to a canonical concept library (Section 4.2.2), and from the library to a trained concept annotator that can label arbitrary text (Section 4.2.3); each stage is validated under a common evaluation framework introduced at the start of Section 4.2. We next describe the hierarchical taxonomy imposed on the library (Section 4.3) and a human interpretability study validating that our concepts are human-meaningful rather than merely LLM-fluent (Section 4.4). Finally, we describe how the trained annotator is applied to the full pretraining corpus (Section 4.5) and how the resulting annotations and embeddings are indexed for test-time training data attribution (Section 4.6).

4.1 Motivation: No human-interpretable concept library exists at scale

Before building our own library, we surveyed existing approaches for large-scale concept extraction. Broadly, existing concept libraries fall into three categories.

- **Word-based concept dictionaries**. Luo et al. (2024) construct a 40,000-item concept dictionary by selecting the most frequent words from the Brown Corpus (Francis and Kučera, 1967) and prompting GPT-4 (Achiam et al., 2023) to generate sentences illustrating each word. Single words, however, cannot capture the higher-level abstractions, multi-sentence topics, or domain-specific ideas that appear throughout pretraining datasets.
- **Activation-derived, unsupervised concepts**. These approaches extract concepts from open-weight model activations, for example, directions discovered via sparse autoencoders (SAEs) (Bricken et al., 2023b). Activation-derived concepts are not constrained to be human-meaningful; a direction in activation space may be statistically salient without corresponding to anything a person would recognize as a coherent idea. There is also no guarantee that these directions will correspond to topics that are expressed in the corpus that the interpretable language model will be trained on (Leask et al., 2025; Korznikov et al., 2026).
- **Narrow domain libraries.** Some concept sets target specific tasks such as sentiment analysis or toxicity detection. For instance, Sun et al. (2025) define concepts using ChatGPT for SST2, Yelp Polarity, and AGNews, specifying categories like world, sport, business, and technology news for AGNews. While these libraries offer high-quality labels, they are far too narrow to supervise models trained on diverse corpora spanning web text, code, mathematics, and scientific writing.

Desiderata for an ideal concept library. We posit that a concept library suitable for supervising language model pretraining at scale must satisfy five properties. The library must...

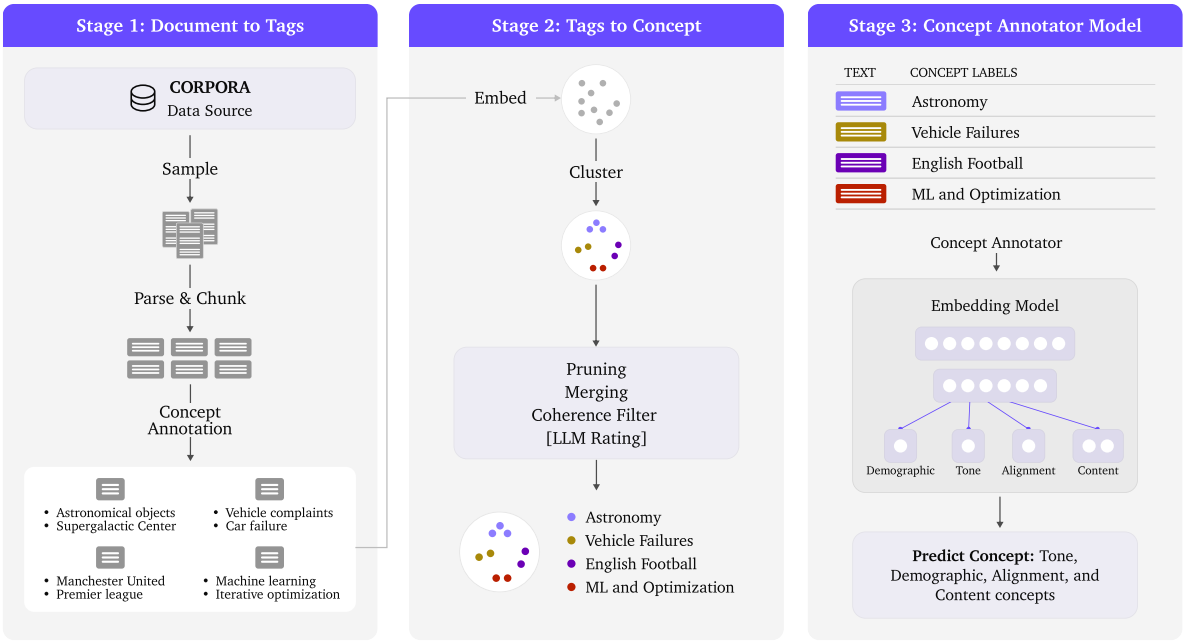

Figure 4. Overview of the Atlas three-stage annotation pipeline: documents are chunked and tagged (Stage 1), tags are clustered and canonicalized into a concept library (Stage 2), and a concept annotator model is trained for scalable text annotation (Stage 3).

1. *Multi-scale*: cover both high-level themes (e.g., "machine learning") and fine-grained units (e.g., "gradient clipping for recurrent networks").
2. *Localizable*: be applicable to spans within a document, enabling sub-document-level supervision and control.
3. *Stylistically expressive*: capture attributes like document tone, formality, or even communicative intent, not restricted to semantic categories.
4. *Representative*: possess the breadth necessary to cover the true distribution of large-scale pretraining corpora across web text, code, mathematics, and scientific writing.
5. *Human-meaningful*: comprise concepts that people—given expertise appropriate to the topic—can understand and would want to use when assessing a language model's behavior.

The last desideratum deserves emphasis. It is tempting to define "human-meaningful" as "labeled by a human" or "labeled by an LLM in a way that sounds reasonable". Neither of these is sufficient. It is possible for an LLM to produce a fluent label for even semantically incoherent document groupings. A human can rate a label as "reasonable" without checking whether they would have arrived at the same label independently. In Section 4.4, we describe a direct test that enables us to assess the correspondence between human-generated labels and LLM-generated labels.

4.2 Atlas: Documents to concepts

Atlas is a three-stage pipeline. Stage 1 samples documents, chunks them into local semantic units, and extracts a high-recall pool of free-form tags. Stage 2 clusters these tags, filters low-quality tag clusters, labels them, and deduplicates them into a canonical library of human-understandable

concepts. Stage 3 trains a concept annotator model that can assign concepts from the library to arbitrary text, enabling annotation of the full training corpus. We describe each stage in turn.

Evaluation overview. The evaluation of Atlas must answer two questions. The first is an engineering question: does the pipeline produce high-quality annotations at each stage? The second is a scientific question: are the resulting concepts genuinely human-interpretable, or merely LLM-fluent? For the engineering question, we use a single framework throughout the pipeline and report results alongside each stage below. A sampled annotation—a raw tag from Stage 1, a canonical concept from Stage 2, or a predicted label from the Stage 3 annotator—is scored against its text chunk on a 1–5 relevance scale, by an LLM judge at scale and by human annotators on smaller matched samples. A score of 2 or higher counts as successful: the tag or concept is at least minimally present in the chunk. This threshold is intentionally permissive, reflecting the high-recall goal of the pipeline and the fact that minor tags can capture fine-grained contextual details without being the dominant topic. We therefore report both success rates and full score distributions, visualized as histograms of per-chunk and per-tag (or per-concept) average scores. On the matched samples, human ratings track the LLM judge but run roughly 0.4 points higher, so the LLM-judged scores reported below are, if anything, conservative.

The scientific question cannot be settled by an LLM judge: an LLM can rate an LLM-generated label as coherent without any guarantee that a human would recognize the same concept. We address it separately with a two-phase human study in Section 4.4, testing whether the concept names are human-interpretable rather than merely fluent.

4.2.1 Stage 1: Corpus sampling, chunking, and documents to tags

Stage 1 constructs the high-recall tag pool from which the concept library is later derived. We begin with a representative sample of our pretraining and midtraining mixtures, split documents into local semantic chunks, prompt an annotator model to assign structured free-form tags to each chunk, and validate the resulting tags. At this stage, tags are not canonical concepts: they are short words or phrases that may be redundant, overlapping, or overly specific. This is deliberate. Stage 1 is optimized for recall; Stage 2 handles clustering, filtering, labeling, and deduplication.

Throughout this section, we distinguish between *tags* and *concepts*.

> A *tag* is a short free-form string assigned to a specific chunk of text during Stage 1 annotation.
>
> A *concept* is a canonical human-interpretable idea produced by clustering, filtering, labeling, and deduplicating many related tags. Each concept has a name, description, and supporting evidence from its associated tags.

Tag assignment is an empirical output of a high-recall annotation process: a tag either was or was not assigned to a chunk, but unassigned tags are not treated as negatives. Concept assignment is a calibrated library-level prediction problem: once tags have been canonicalized into concepts, we evaluate whether a named concept is genuinely present in a chunk and whether humans recognize the concept from its supporting evidence. Concepts are necessarily less fine-grained than raw tags, and much of the calibration work in Atlas concerns whether a named concept should be assigned to particular chunks of text.

Source mixture. We sample from five major document categories: *web text*, using a deduplicated version of DCLM from Zyda-2 (Li et al., 2024; Tokpanov et al., 2024); *general academic knowledge*, including peS2o (Soldaini and Lo, 2023), arXiv (Weber et al., 2024), and Wikipedia and Wikibooks

Chunks	Structured Tags
The game increases the chances of finding a Shiny Pokemon by generating extra personality values in an attempt to find one that results in a Shiny Pokemon, with the number of attempts depending on the size of the current streak. For every Pokemon added to the streak up to 20 Pokemon, the game will make two extra attempts to find a Shiny personality value; i.e., the number of attempts at any given point in the streak is $1 + 2*$ streak_size, and caps at a maximum of 41 attempts when the streak is at least 20 Pokemon long.	**main:** gaming, mechanics, video-games, Pokemon, shiny-mechanics **purpose:** informational-guide, explanatory, how-to **tone:** neutral, informative, technical **minor:** gaming-progress, streaks, gaming-algorithms, random-generation
In Going for the Gold!, Ash and his friends met a fisherman named Rodman, who was trying to fish up a Shiny Magikarp with a Magikarp-shaped lure. Ash, Serena, and Clemont decided to try fishing too, with Ash giving the inexperienced Serena instructions of how to do it correctly. While fishing, Serena hooked up a Corsola, which she tried to battle with her Fennekin, but it simply hid itself behind Serena when Corsola tried to use Water Gun on it, causing the Coral Pokémon to get away.	**main:** fiction, media-franchise, anime, pokemon, adventure **purpose:** narrative, storytelling, character-development **tone:** casual, descriptive, lighthearted **minor:** fishing-techniques, lure-use, pokemon-battles, strategy-mistakes

Table 3. A webtext document about *Pokemon* split into two chunks, with its domain-specific, hierarchical structured tags.

(from Dolma 1.7 (Soldaini et al., 2024)); *mathematics*, including Dolmino-math, GSM8K (Cobbe et al., 2021), OpenWebMath (Paster et al., 2024b), and Algebraic Stack (Azerbayev et al., 2023a); *code*, using StarCoder (Li et al., 2023); and *question–answer exchanges*, using FLAN v2 (Longpre et al., 2023). In total, the sample contains 6.6 million documents balanced across these categories. This mixture was chosen not as a benchmark distribution, but as a substrate for concept extraction: it spans multiple writing styles, knowledge domains, and levels of conceptual density.

Chunking. We annotate at the chunk level rather than the document level. Long documents often contain several unrelated ideas, and whole-document annotation tends to collapse them into coarse summaries. A chunk is a short, semantically coherent span, created by detecting sentence boundaries with the BlingFire sentence splitter (Microsoft, 2019) and concatenating consecutive sentences until a domain-specific token threshold is reached. We use a threshold of 150 tokens for web text and general documents and 256 tokens for mathematics and code, where a single mathematical or algorithmic idea often requires more context. Sentences exceeding the threshold are retained as single chunks; extremely long sentences above 50,000 tokens are dropped as malformed or uninformative for local annotation. This process yields 44 million chunks from the 6.6 million sampled documents.

Chunking at this granularity is essential: it makes tags local to the semantic units they describe, rather than global summaries of entire documents.

Domain-aware annotation schemas. Different domains express conceptual structure differently. A Wikipedia article, a mathematical proof, a Stack Exchange question, and a block of Python code call for different annotation fields. We therefore designed domain-specific structured tag schemas, each containing 4–6 fields tailored to the content type. Each field elicits a hierarchy of tags from broad to narrow, and together the fields produce on average 10–15 tags per text chunk.

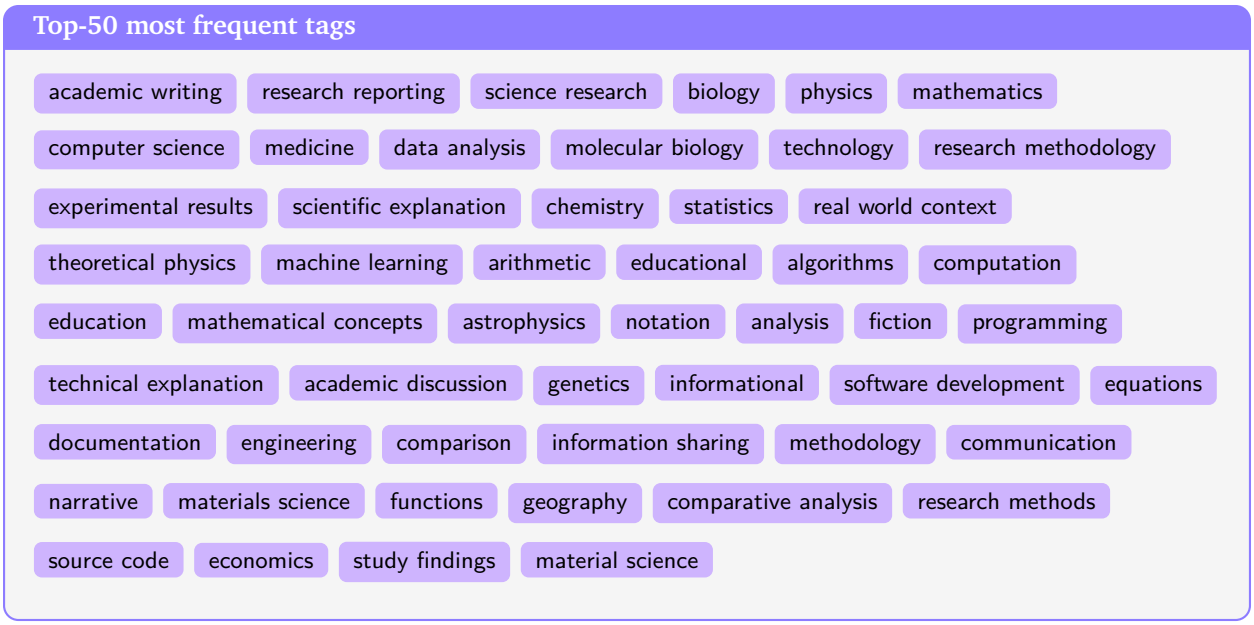

Figure 5. The 50 most frequent raw tags produced by Stage 1. These tags illustrate the high-recall, redundant, non-canonical tag space before Stage 2 clustering and deduplication.

For each domain, the schema separates several complementary views of a chunk: what the chunk is about, what role it plays, and how it is expressed. For web text, fields include topic, communicative purpose, tone or style, and secondary entities or events; Table 3 shows an example web text document split into chunks with its domain-specific structured tags. For mathematics, fields capture the mathematical object or topic, problem type, solution or proof technique, notation conventions, and difficulty. For code, fields capture programming language, design pattern, algorithmic structure, software-engineering role, and documentation style. The schemas are deliberately overcomplete: a single chunk may receive several overlapping tags, because Stage 1 is optimized for recall before Stage 2 clusters the tag space into canonical concepts.

Annotator model selection. We evaluated several open-weight models for structured annotation, including Phi-family models (Abdin et al., 2024), `Qwen2.5-7B` (Qwen et al., 2025), and `Mistral-Small-3.1-24B-Instruct` (Mistral AI Team, 2025). The choice of annotator model is constrained by a practical requirement: reliability at scale matters more than raw capability. Even a 1% schema deviation rate across 44 million chunks produces nearly half a million unusable annotations.

We found that Phi-family models could output consistently structured annotations but suffered from repetition collapse and degenerate loops, making them unsuitable for long-running annotation jobs. `Qwen2.5-7B` performed better semantically but less predictably syntactically, often producing the wrong number of fields or responses that were difficult to parse. `Mistral-Small-3.1-24B-Instruct` consistently adhered to the structured format, avoided repetition collapse, and maintained stable behavior across millions of prompts. Mistral was the smallest model that satisfied our formatting and reliability constraints.

Output. Stage 1 produced nearly 44 million annotated chunks, from which we extracted approximately 500 million tags. After tag deduplication, 14 million unique tags remained, of which approximately 1 million appeared more than 15 times. The most frequent raw tags, shown in Figure 5, provide a sanity check on this high-recall regime: the tag pool captures broad recurring content and

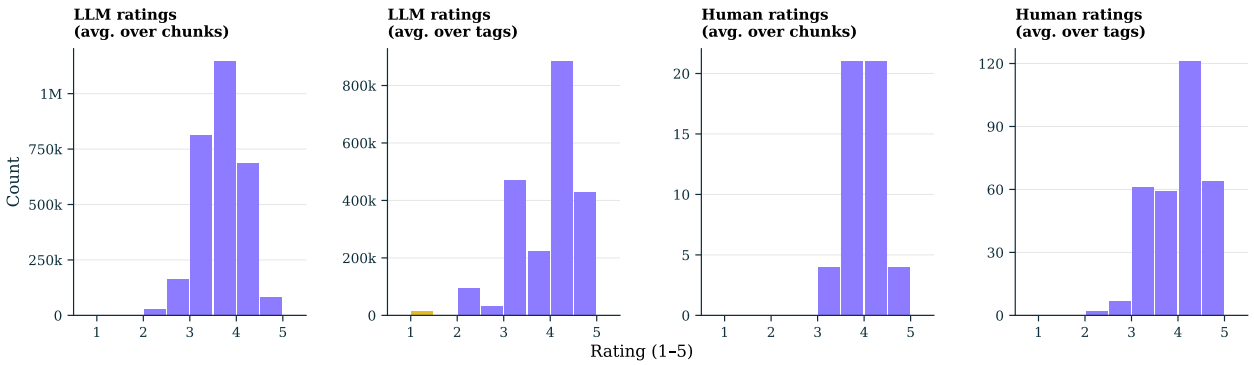

Figure 6. Stage 1 tag validation scores. Distributions of average tag-relevance ratings on a 1–5 scale: per-chunk averages (left column; each chunk's tag ratings averaged) and per-tag averages (right column; each unique tag's ratings averaged over the chunks it appears in), as rated by the LLM judge over 2.93M sampled chunks (top row) and by three human annotators on a 50-chunk sample (bottom row).

stylistic patterns, but remains redundant and non-canonical before Stage 2 clustering. This raw tag space is intentionally redundant and noisy.

> At this stage of the pipeline, missing a relevant concept is more damaging than producing overlapping or synonymous tags: synonymy and over-specificity can be removed by clustering, while absent concepts cannot be recovered downstream.

The purpose of Stage 1 is therefore to cast a wide semantic net. Stage 2 consolidates this tag space into a large concept library.

Using the evaluation framework described at the start of this section, an LLM judge scored 21.3 million (tag, chunk) pairs over a sample of 2.93 million chunks (2.15 million unique tags), with human annotators rating a 50-chunk subsample; the judging prompt is given in Part III. Individual tag ratings average 3.62, and 97.5% score at least 2.

> Across all domains, nearly every tag is at least minimally present in its chunk: per-chunk average scores center near 3.6, with two-thirds of chunks falling between 3.2 and 4.0 and 99.9% averaging at least 2 (Figure 6).

4.2.2 Stage 2: Tags to concepts

Stage 2 transforms nearly 14 million noisy, free-form tags into a coherent library of over $33{,}000$ human-understandable concepts. The process embeds tags into a semantically meaningful space, clusters them, filters low-coherence clusters, labels each surviving cluster, and merges near-duplicate concepts through graph-based deduplication. We describe each step in turn.

Tag normalization and embedding. Raw tags produced by an LLM from Stage 1 are highly variable: minor formatting differences—hyphens, slashes, whitespace, punctuation artifacts—can split semantically identical tags into separate strings. We normalize all tags into a standardized form before embedding, e.g., `astronomical-objects` $\rightarrow$ `astronomical objects`, `climate-change-adaptation` $\rightarrow$ `climate change adaptation`. Each unique tag is then embedded into an n-dimensional ($n = 768$) space using the `all-mpnet-base-v2` sentence embedding model (Sentence Transformers Team,

2021). This choice is empirical: in a comparison against `Qwen3-Embedding-0.6B` (Zhang et al., 2025), the former produced slightly higher-quality clusters, as measured by standard quantitative clustering metrics—Silhouette score (Rousseeuw, 1987), Davies–Bouldin index (Davies and Bouldin, 1979)—together with the LLM-judge cluster coherence evaluation described below. In total, this step produces nearly 14 million vectors, one per unique tag.

Clustering. With all tags embedded in a common semantic embedding space, we cluster them into groups representing candidate concepts; for instance, *missing pet*, *missing pet incident*, *missing dog* should fall in a single cluster. We use k-means, implemented in the FAISS library (Johnson et al., 2019) for GPU-accelerated efficiency over tens of millions of vectors.

The number of underlying semantic clusters is unknown, so we sweep a wide range of cluster counts: $k \in \{100, 500, 1\text{k}, 10\text{k}, 20\text{k}, 30\text{k}, 50\text{k}, 80\text{k}, 100\text{k}\}$. Small k produces large, diverse clusters that conflate unrelated concepts; large k produces tight clusters but risks fragmenting genuinely related concepts or introducing clusters based on noise. We evaluate each k with the Silhouette score, which rewards clusters that are internally cohesive and well separated, the Davies–Bouldin index, which penalizes clusters whose internal scatter is large relative to their separation, alongside the LLM-based cluster coherence evaluation described next.

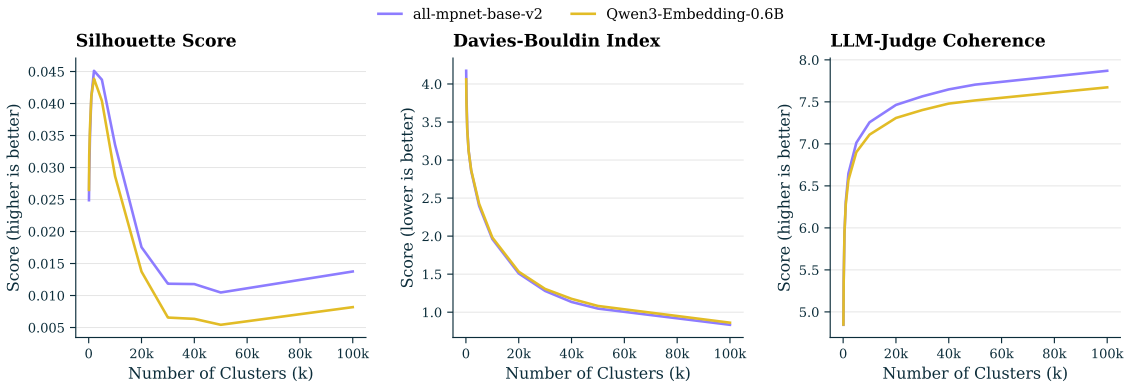

Figure 7. Comparison of `all-mpnet-base-v2` and `Qwen3-Embedding-0.6B` embeddings for clustering ∼14M LLM-generated tag annotations via the k-means clustering algorithm. We evaluate Silhouette score (left plot), Davies–Bouldin index (middle plot), and coherence score (right plot) on randomly sampled cluster members. As k increases, `all-mpnet-base-v2` consistently yields more coherent clusters than `Qwen3-Embedding-0.6B`.

LLM-based cluster coherence evaluation. The two standard metrics capture broad trends across large changes in k but do not reliably distinguish nearby values, so we complement them with a direct semantic coherence evaluation of every cluster. For each cluster, we sample three strata of tags—core (closest to the centroid), random (uniform), and edge (farthest from the centroid)—group them into sets of ten and query an LLM, `Mistral-Small-3.1-24B-Instruct`, to score the semantic coherence of each set on a 1–10 scale, yielding a coherence score per stratum.

Coherence improves steadily with k and plateaus around $k = 40{,}000$–$80{,}000$ as shown in Figure 7; beyond this range, clustering computational cost rises while semantic gains diminish, as larger k begins to split coherent groups across multiple clusters. We therefore set the initial number of clusters to $k = 80{,}000$.

Quality filtering. Not every cluster corresponds to a coherent concept; some arise from incidental lexical overlap rather than shared meaning. We retain only clusters that clear strict per-stratum

coherence thresholds—core ≥ 9, random ≥ 8, and edge ≥ 7. Of the 80,000 clusters, approximately 17,000 fail these criteria and are removed, leaving roughly 63,000 high-quality clusters.

Cluster labeling. Each surviving cluster is converted into a human-interpretable concept. We sample 50–100 representative tags per cluster, weighted by tag frequency, and prompt an LLM, `Mistral-Small-3.1-24B-Instruct`, to generate a concise label (1–6 words) and a one-sentence rich description; the prompt is given in Part III. These labels provide a clean, human-friendly interpretation of what are otherwise dense numerical clusters of tags.

> The LLM *labels* structure that emerged from clustering millions of human-comprehensible tags; it does not fabricate concepts from nothing.

The tags were produced by annotating real text, the clusters capture statistical regularities in those tags, and the LLM assigns a name to each regularity. Whether the resulting names are genuinely human-interpretable is an empirical question we address in Section 4.4.

Graph-based cluster deduplication. The labeled concepts, now numbering $62{,}000$, still contain substantial redundancy: multiple clusters may express the same underlying idea, differing only in granularity or phrasing. We address this redundancy by iterative graph-based concept merging.

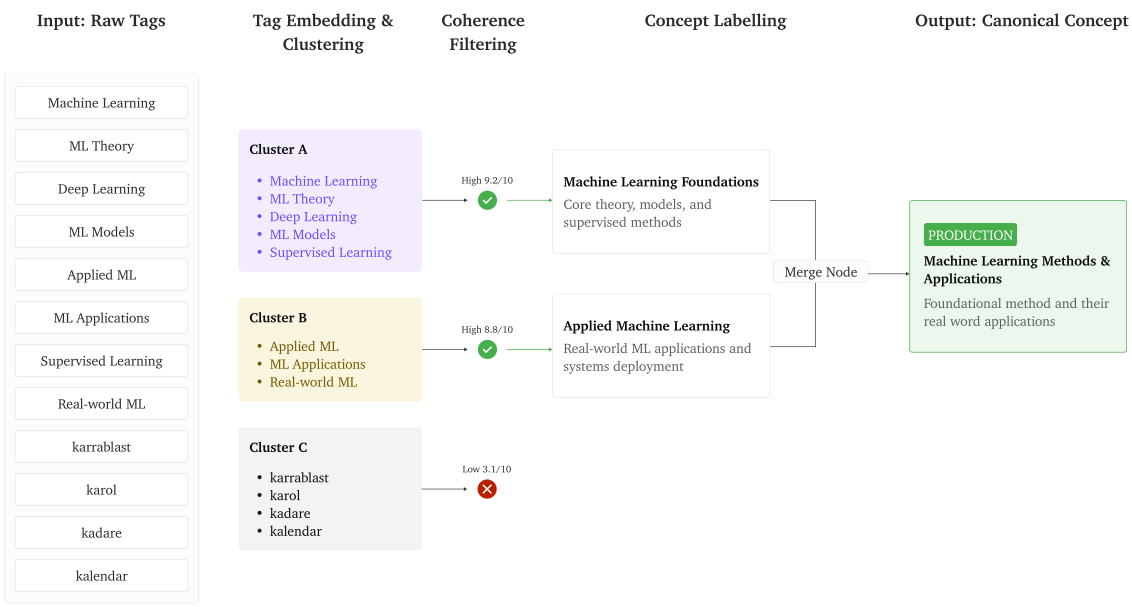

Figure 8. Stage-2 transformation of noisy LLM tags into a canonical concept: raw tags are embedded and clustered, incoherent clusters are filtered out, each surviving cluster is labeled into a single concept (name with italic description), and semantically adjacent concepts are merged by cosine similarity into one canonical entry.

We embed each concept's label and description with `Qwen3-Embedding-0.6B` into a semantic n-dimensional vector space, then connect each concept to its m-nearest neighbors, retaining edges with cosine similarity above a certain threshold τ. This produces an undirected similarity graph whose nodes are concepts and whose edges connect near-duplicates. We partition the graph with Louvain-community detection (Blondel et al., 2008) to identify groups of related concepts, treat

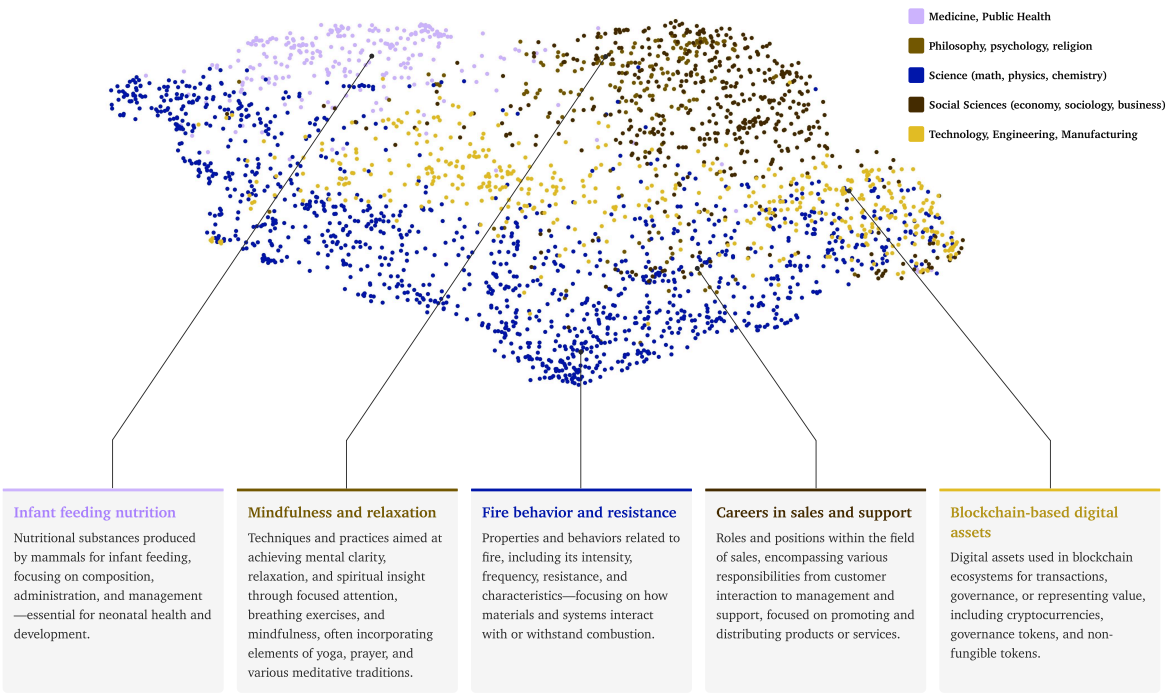

Figure 9. UMAP projection of a subsample of concept embeddings across five taxonomy groups. The cards showcase representative concepts from distinct regions of the embedding space to illustrate local semantic clusters.

each community as a candidate merge set, and prompt an LLM to regenerate a single unified label and description for each set. A single pass reduces the library to approximately 39,000 concepts. We then re-embed all concepts and repeat—rebuilding the graph and re-running community detection at each iteration—halting after two to three iterations once an iteration merges only a negligible number of concepts. This leaves over $33{,}000$ concepts. We chose $m = 20$, $\tau = 0.95$ for first iteration, reduced $\tau = 0.9$ for second pass, and then $\tau = 0.85$ for third pass based on empirical evidence.

Output.

> The final canonical concept library contains $33{,}732$ concepts spanning science, technology, philosophy, medicine, law, and other domains.

Some conceptual overlap and hierarchy are unavoidable—real-world knowledge is not cleanly partitioned—but the deduplicated library strikes a practical balance between granularity and clarity. We visualize the concept embeddings in Figure 9 using UMAP (McInnes et al., 2018), a scalable nonlinear dimension-reduction method for visualization and manifold learning.

Validation. Using the same framework, an LLM judge scored the concepts associated with each of the same 2.93 million chunks via their Stage 1 tags—12.2 million (concept, chunk) pairs covering nearly the entire library—with human annotators again rating a 50-chunk subsample (Figure 10). Individual ratings average 3.50, with 98.0% scoring at least 2; per-chunk averages center near 3.5, with two-thirds of chunks between 3.2 and 4.0 and 99.7% averaging at least 2.

Per-concept average scores are notably uniform across the library (mean 3.51, standard deviation 0.41): concept quality is consistent rather than driven by a head of frequent, well-represented concepts.

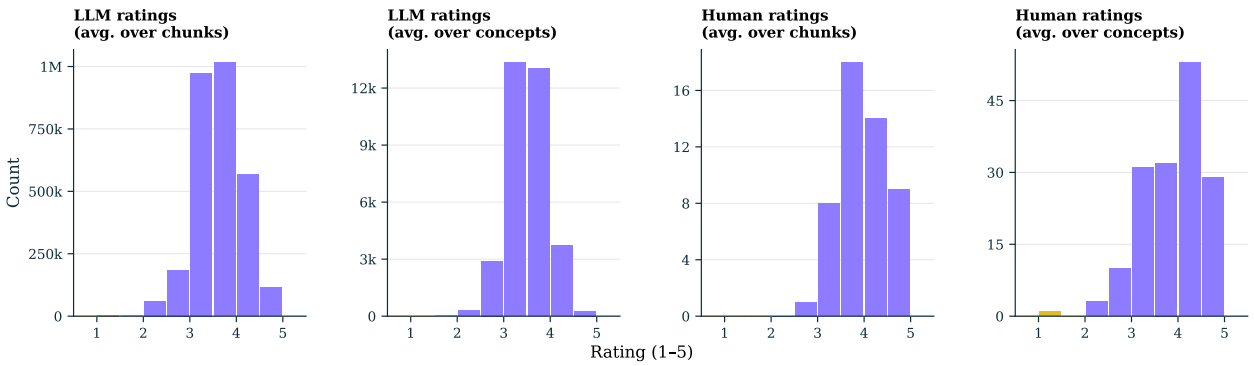

Figure 10. Stage 2 concept validation scores. Distributions of average concept-relevance ratings on a 1–5 scale: per-chunk averages (left column) and per-concept averages (right column; each concept's ratings averaged over the chunks it is assigned to), as rated by the LLM judge over 2.93M chunks (top row) and by human annotators on a 50-chunk sample (bottom row).

4.2.3 Stage 3: Concept annotator model

The outcome of Stages 1 and 2 is a large canonical library of content concepts, defined over the sampled corpus. Stage 3 trains a scalable annotator that maps an arbitrary input chunk to a set of target labels, allowing us to annotate the full 1.5 trillion-token pretraining corpus. The annotator predicts over a combined target space consisting of the Atlas-derived content concepts together with several lower-cardinality auxiliary domains: tone, demographic-reference, and alignment-relevant labels.

Training targets. The Stage 3 training targets come from two sources. The first source is the content-concept library produced by Stages 1 and 2. For each chunk in the Stage 1 annotated sample, raw tags are mapped through the Stage 2 clustering and deduplication pipeline to the final content-concept IDs. Thus, if a tag assigned to a chunk belongs to a cluster that is ultimately canonicalized as concept c, then c is inherited as a positive content label for that chunk. These inherited labels provide positive-only supervision: concepts not assigned to a chunk are treated as unlabeled rather than verified negatives.

Rather than train directly on all 44 million Stage 1 annotated chunks, we construct a long-tail-enriched training reservoir. The reservoir is selected to improve coverage of rare content concepts by amassing a minimum number of examples per concept. This procedure yields roughly 2.9 million chunks. The resulting reservoir is still highly imbalanced—frequent concepts can appear tens of thousands of times—but it gives substantially better coverage of the tail than uniform sampling from the full Stage 1 sample. We train the annotator on approximately 2.3 million reservoir chunks and reserve approximately 597,000 chunks for held-out evaluation.

The second source of supervision consists of fixed candidate vocabularies for auxiliary lower-cardinality domains. These domains are not produced by the Stage 2 clustering procedure. Instead, we define finite candidate lists for tone, demographic references, and alignment-relevant categories, and annotate chunks using the same LLM-based annotation infrastructure with separate lists for each domain. Given a chunk and a candidate list, the annotator is asked which categories are rel-

evant to the text. Selected categories are used as positive labels for the corresponding auxiliary heads. The final prediction vocabulary contains $33{,}732$ labels in total, dominated by approximately $33{,}606$ content concepts, together with roughly 80 tone labels, 38 demographic-reference labels, and 8 alignment-relevant labels.

Design constraints. A few key requirements shaped the annotator model design.

- First, we want a single model that can handle both the high-cardinality Atlas-derived content concept library and the lower-cardinality auxiliary domains described above, rather than maintaining separate classifiers with separate encoders, thresholds, and inference logic.
- Second, in Stage 2, we found that a simple k-nearest neighbor classifier over the concept embeddings, performs surprisingly well on content-label prediction, especially for frequent, well-represented concepts. We want a flexible architecture and loss function family that retains this simplicity, but incorporates learned signals for rare classes.
- Third, our training data contains only positive labels for each concept domain since we did not do negative annotation: each chunk is annotated with the concepts it *has*, never with the concepts it explicitly does not have. This positive-unlabeled (PU) supervision regime requires careful treatment of loss functions and evaluation metrics, so we will restrict ourselves to settings that make it flexible to incorporate such design constraints (Denis, 1998; De Comité et al., 1999; Denis et al., 2005; Kiryo et al., 2017).

Architecture. The annotator uses the `Qwen3-Embedding-0.6B` model as a shared encoder, producing a pooled embedding for each input chunk. This embedding feeds into a lightweight MLP trunk (LayerNorm, ReLU, Dropout, projection to 1024 dimensions) shared by the auxiliary prediction heads. The content head follows a different path: it computes dot products between the encoder output and a matrix of content-concept embeddings, one per Atlas-derived content concept. This design preserves the KNN-like signal of direct embedding similarity while allowing learned improvements. The tone, demographic-reference, and alignment-relevant heads use simpler linear classifiers, which suffice for their lower-cardinality fixed vocabularies.

Loss. We combine two loss functions. The first is masked binary cross-entropy, applied only to positions where targets are non-zero, which handles the positive-only supervision. The second is a non-negative PU loss, a PU-compatible objective that penalizes overconfident predictions on unlabeled classes and stabilizes learning on long-tail distributions. Rare concepts are further supported by a rarity-weighted sampling scheme that boosts underrepresented labels during training.

Evaluation. Standard classification metrics—precision, recall, PR-AUC—are problematic under PU learning, because any unlabeled example predicted as positive is counted as a false positive even if the prediction is correct. This systematically deflates precision. We track these metrics for monitoring relative improvement across training runs, but we do not rely on them for absolute quality judgments. Instead, we rely on the LLM and human evaluation framework described at the start of the section.

The annotator is trained on 2.3 million of the sampled chunks; we evaluate its predictions on the remaining 597,000 held-out chunks, scoring 4.93 million predicted (concept, chunk) pairs with the LLM judge (Figure 11). Predicted-concept ratings average 2.94, with per-chunk averages clustering between 2.5 and 3.5 (median 3.0); 95.5% of per-chunk averages are at least 2, and per-concept averages center near 3.3.

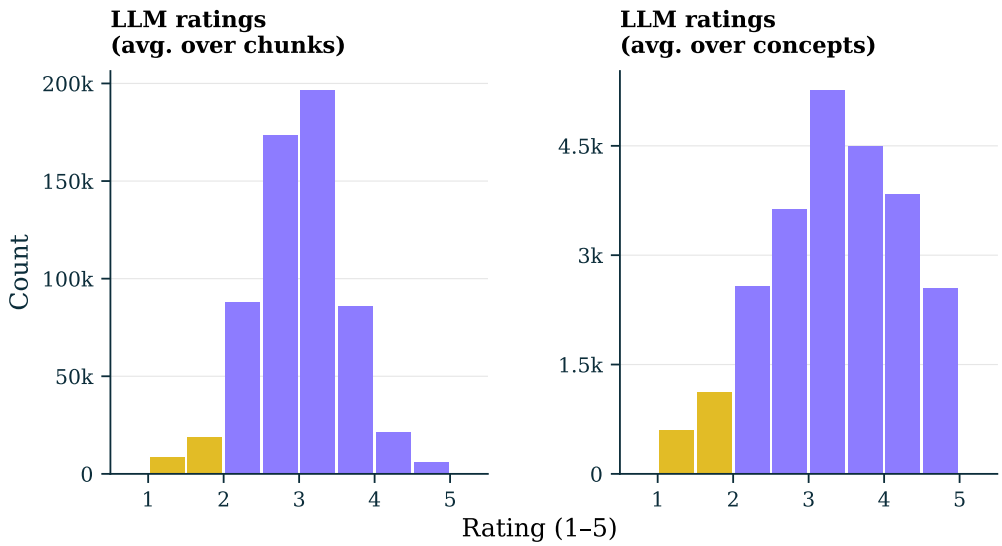

Figure 11. Stage 3 annotator validation on held-out chunks. Distributions of average predicted-concept relevance ratings from the LLM judge: per-chunk averages (left) and per-concept averages (right).

The annotator thus assigns acceptable concept sets to the overwhelming majority of held-out chunks, though its predictions score lower and are more dispersed than original tag and concept ratings—plausibly reflecting the positive-unlabeled training regime and the per-domain prediction caps.

4.3 Concept taxonomy

We organized the 33,732 canonical concepts into a hierarchical taxonomy derived from the Library of Congress Classification (LCC), an established bibliographic classification system maintained by the Library of Congress (Library of Congress, 2023). We used the `agentlans/library-classification-systems` dataset (Tseng, 2024), which provides a machine-readable outline of LCC entries with parent–child links. The resulting taxonomy maps the concepts onto roughly 2,600 occupied nodes within the full 6,517-node LCC outline, with populated paths reaching depth 9 (Figure 12).

Concept distribution across taxonomy. Science (Q) dominates with 38% of concepts, followed by Technology (T) at 15%, Social Sciences (H) at 15%, and Medicine (R) at 8%. All 20 root branches are represented to varying degrees. Within the Science branch, mathematics and physics subcategories are particularly prominent: Mathematics (QA) subdivisions such as Analysis, Geometry, and Algebra account for over 3,700 concepts combined, while Physics (QC) areas such as Atomic and Molecular Physics contribute over 660 concepts.

The distribution in Figure 13 reflects the composition of the underlying pretraining corpus rather than an editorial choice about which domains matter. The taxonomy provides a structured framework for analyzing concept coverage across knowledge domains and, as we describe below, enables stratified evaluation of concept quality.

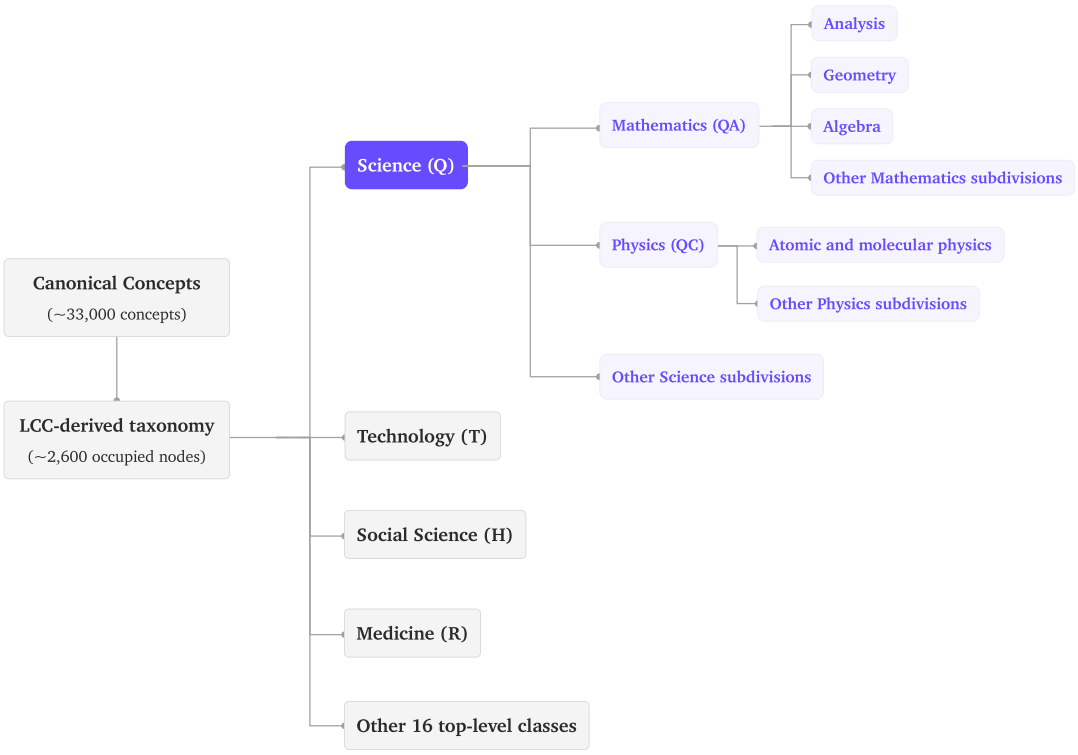

Figure 12. Schematic of the LCC-derived taxonomy used to organize the Atlas concept library. The 33,732 canonical concepts are mapped onto approximately 2,600 occupied taxonomy nodes. The figure shows the root structure and expands the Science (Q) branch to illustrate how top-level classes decompose into more specific areas such as Mathematics (QA), Physics (QC), and their subdivisions; other branches are collapsed for readability.

4.4 Human interpretability of the concept library

The validations above show that Atlas is internally consistent: tags are relevant to chunks, clusters are semantically coherent, and the trained annotator assigns concepts to held-out text that an LLM judge scores as present. This does not, by itself, show that the final concept names are human-interpretable. Since the Stage 2 concepts are labeled by an LLM, there is a specific failure mode we must rule out: the LLM might assign fluent names to statistical patterns that do not correspond to concepts humans recognize. For example, a cluster whose tags co-occur for lexical rather than semantic reasons can still receive a confident, plausible-sounding name—and an LLM judge shown the same evidence may rate that name as coherent—without any person being able to independently arrive at, or even recognize, the underlying concept.

We therefore evaluate two claims. First, the evidence associated with a concept must itself contain recoverable semantic structure. Second, the Atlas label—generated by an LLM in Stage 2—must name that structure at least as well as labels generated independently by humans.

Lifted-word evidence. For each concept c, we construct a list of *lifted words*: lemmatized words appearing in the text chunks assigned concept c, ranked by how strongly they are associated with c

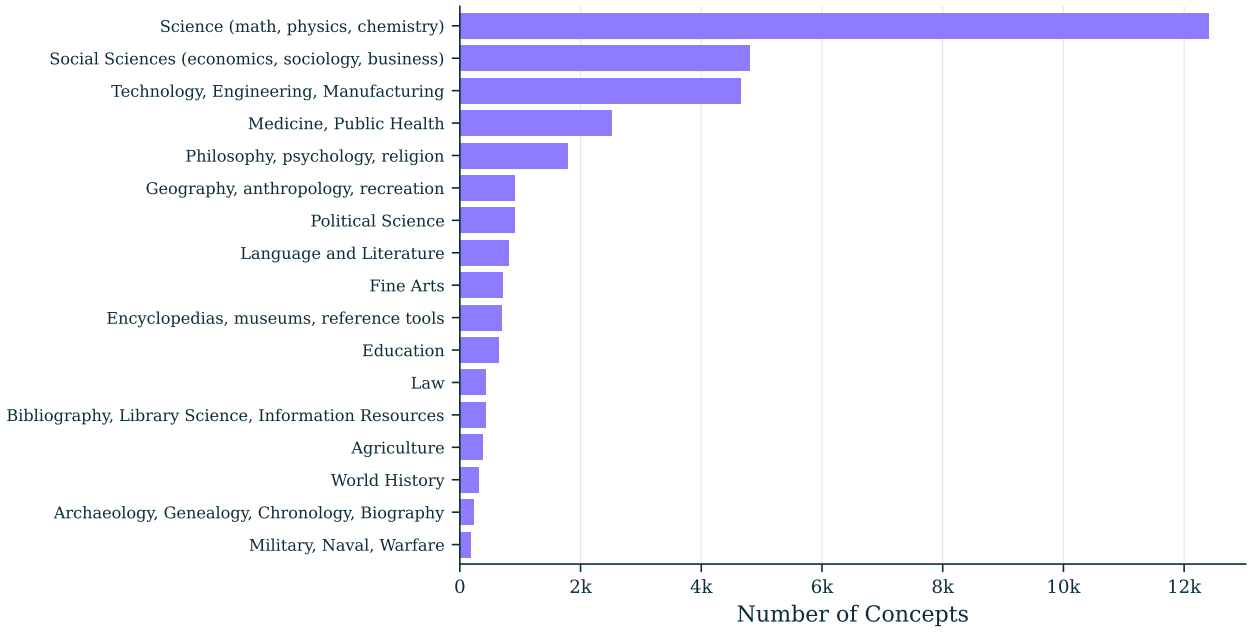

Figure 13. Distribution of over 33,000 concepts across the top-level LCC classes and notable subclasses. Science (Q) accounts for the largest share, followed by Technology (T), Social Sciences (H), and Medicine (R); all root classes are represented to varying degrees. The distribution mirrors the composition of the pretraining corpus rather than a curatorial choice.

relative to their background frequency across all chunks,

$$\text{lift}(w, c) = \frac{P(w \mid c)}{P(w)},$$

with a minimum-support filter to remove idiosyncratic rare words. Lifted words provide a readable, corpus-level form of concept evidence. They also instantiate the more general evaluation problem used elsewhere in the paper: given only the words or tokens statistically associated with a concept, can a labeler recover a human-meaningful name?

Two-phase human study.

We ran a two-phase human study. In Phase 1, human annotators saw only the lifted words for a concept. Each annotator wrote a name or short phrase for the lifted-word list and rated, on a 1–5 coherence scale—from 1 (the words form no recognizable concept) to 5 (the words clearly correspond to a single recognizable concept)—whether the words formed a recognizable concept at all. This generation task is deliberately stricter than asking humans to approve a provided label: annotators can mark the evidence as noisy rather than being forced to accept a fluent name.

In Phase 2, annotators performed blind comparative scoring. For each concept, we assembled a candidate set containing the Atlas label generated in Stage 2, two human labels written for the same concept by other annotators in Phase 1, a *taxonomy distractor* from a nearby but distinct concept in the taxonomy, and an *embedding distractor* from a different concept with a nearby label embedding. In a small number of cases where a second human label was unavailable, we substituted a deliberately generic filler label as a floor control: a label expected to fit poorly, confirming that raters used the low end of the scale. Annotators rated how well each candidate name fit the same lifted-word list, again on a 1–5 scale. Candidate order was randomized, annotators were blind to label provenance, and no annotator scored a label they had written.

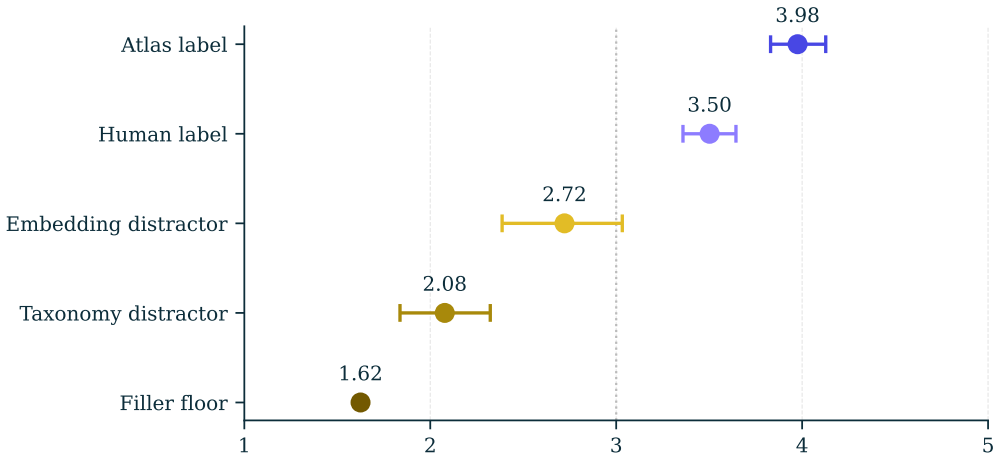

Figure 14. Phase 2 human-evaluation fit scores by candidate type. Human annotators rated, on a 1–5 scale, how well each candidate name fit the same lifted-word evidence for a concept. Points show mean fit scores and horizontal bars show 95% confidence intervals; the dotted vertical line marks the neutral midpoint of the scale. Atlas labels score comparably to or above independently generated human labels, and both are clearly separated from embedding, taxonomy, and filler distractors, indicating that raters were not merely assigning high scores to any plausible label.

Do lifted words contain recognizable structure? Yes, although not in every case. We sampled 100 concepts stratified by top-level taxonomy branch: ten concepts from each of the nine largest branches and ten from the aggregated remainder. Phase 1 collected 303 named-concept responses from 9 human annotators, with a median of 3 annotators per concept. Human-written names averaged 4.1 words.

The mean Phase 1 coherence score was 3.52. Annotators judged 55% of responses to form a recognizable concept (≥ 4), rated 27% as borderline ($= 3$), and flagged 17% as incoherent or noisy (≤ 2; on this coherence scale, unlike the relevance scale used in the pipeline validations, low scores indicate unrecognizable or noisy evidence).

> Thus the premise is empirically non-trivial: many lifted-word lists contain semantic structure that humans can independently recognize, while a minority remain ambiguous or noisy.

Do Atlas labels name that structure? Yes. Phase 2 collected 1,025 individual candidate-name ratings: 205 scoring records over 34 concepts from 8 annotators. In blind scoring, annotators sharply separated real labels from distractors, validating the task itself (Table 4).

> Atlas labels and human-generated labels both scored far above taxonomy and embedding distractors, showing that raters were not merely assigning high scores to any plausible phrase.

The strongest comparison is between the Atlas label and independently generated human labels. Atlas labels scored higher on average: 3.98 versus 3.50. They also received a top-two rating (≥ 4) 79% of the time, compared with 63% for human labels. In paired comparisons, the Atlas label outscores a human label with probability 0.62 (cluster-bootstrap 95% CI $[0.58, 0.66]$). Additional ordinal mixed-model and robustness analyses are reported in Appendix D.

Interpretation. These results address the central question of validity: humans can infer meaningful

candidate type	mean fit (1–5)	sd	n
LLM (Atlas) label	3.98	1.04	205
human label	3.50	1.17	402
embedding distractor	2.72	1.30	205
taxonomy distractor	2.08	1.11	205
filler (floor)	1.62	1.06	8

Table 4. Phase 2 blind fit scores by candidate type. Human annotators rated how well each candidate name fit the same lifted-word evidence. Counts sum to 1,025 individual ratings.

concepts from lifted-word evidence alone.

> When asked to judge labels blindly, humans rate the Atlas labels (LLM-generated) at least as highly as independently human-generated labels, and far above nearby distractors.

This does not certify every concept in the library individually, but it does show that Atlas is not merely producing LLM-fluent names for arbitrary clusters. On a stratified pilot, the labels are human-recognizable, preferred to strong distractors, and competitive with or better than human-written names.

Connection to known concept alignment. The same principle underlies the known concept alignment metric used in model evaluation (Section 7). Known concept alignment asks whether the word/token evidence associated with a concept from the library—for example, lifted words or the tokens most boosted by a steering vector—matches that concept's name. The human study validates this question as meaningful and supports using an LLM judge as a scalable proxy:

> humans independently recognize the semantic structure in the evidence and endorse the Atlas labels under blind comparison.

The validation chain is therefore: human-recognizable word/token evidence $\rightarrow$ human-endorsed Atlas labels $\rightarrow$ scalable LLM-judged concept alignment.

4.5 Data Processing

The Atlas pipeline described in Section 4.2.1–Section 4.2.3 produces a concept library and a trained annotator model. This subsection describes how we apply the annotator to the full pretraining corpus, producing the tokenized, concept-annotated training data that the Steerling model consumes. The process has three stages: distributed concept annotation with embedding extraction, tokenization with document-structure markup, and retrieval index construction (Section 4.6).

4.5.1 Distributed Concept Annotation

The Stage 3 annotator (Section 4.2.3) must be applied to every chunk in the pretraining corpus—approximately 11 billion text chunks spanning 1.5 trillion tokens. We distribute this workload across GPU nodes using SLURM array jobs, where each task processes a single input parquet file independently.

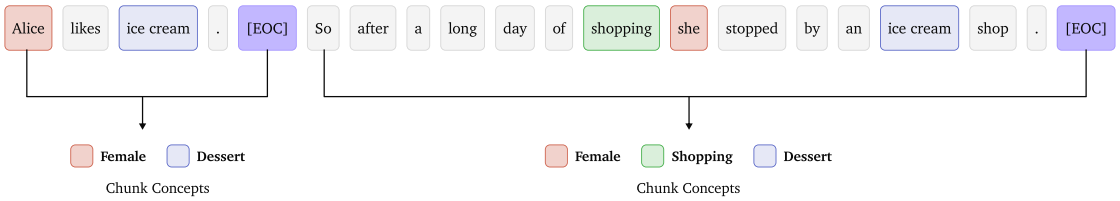

Figure 15. Token stream with chunk-level concept annotations. Each chunk is terminated by an `[EOC]` token and associated with a set of concept labels. Concept supervision is provided at the chunk level; the model learns to localize concepts to individual tokens via OR-aggregation during training.

Input format. Each input parquet file contains pre-chunked text with columns for `chunk_text` (the main content), `data_src`, `doc_id`, `source_doc_id`, `chunk_id`, `doc_end_flag` (marking the final chunk of a document), and `metadata`. The chunking follows the same procedure described in Section 4.2.1: sentence boundaries are detected with BlingFire sentence splitter (Microsoft, 2019), and consecutive sentences are concatenated until a domain-specific token threshold is reached (150 tokens for web text, 256 for mathematics and code).

Forward pass. For each chunk, the annotator's `Qwen3-Embedding-0.6B` encoder produces a pooled 1024-dimensional embedding, which is then passed to the multi-head classifier. The classifier predicts concept activations across all four domains—content, tone, demographic, and alignment—using domain-specific thresholds and per-domain caps on the maximum number of predicted concepts. Each domain's predictions are local indices into that domain's concept vocabulary; these are mapped to global concept IDs via fixed offsets, producing a unified list of concept annotations per chunk.

Embedding extraction. During the same forward pass, we cache the 1024-dimensional encoder embeddings for every chunk, storing them in Zarr arrays at FP16 precision alongside document metadata (`data_src`, `source_doc_id`, `doc_id`, `chunk_id`). These embeddings serve two purposes: they form the basis of the retrieval index constructed in Section 4.6, and they are available for downstream analysis without requiring additional encoder inference. At the scale of the full training corpus, this amounts to approximately 11 billion vectors.

Output format. Each input parquet file produces a corresponding annotated parquet file with the original columns preserved and new columns added: `text` (combined chunk text), `concepts` (list of global concept IDs), and per-domain arrays (`content_global_ids`, `tone_global_ids`, `demographic_global_ids`, `alignment_global_ids`). These annotated parquet files are the input to the tokenization stage.

4.5.2 Tokenization and document structure

The annotated parquet files are tokenized into the token stream consumed by the model during training. We use the `cl100k_base` encoding from tiktoken[1]. We add four special tokens to represent our document and training structure, and reuse the encoding's native `<|endoftext|>` token (ID 100,257) as a document-boundary marker. Including the byte-pair tokens, the encoding's existing special tokens, and our additions, the full vocabulary contains 100,281 tokens.

[1] https://github.com/openai/tiktoken

Token	ID	Role
<|endoftext|>	100257	End of document
[PAD]	100277	Padding
[BOS]	100278	Beginning of sequence
[EOC]	100279	End of chunk (within a document)
[MASK]	100280	Masked position (for diffusion training)

Tokenization procedure. Each chunk's text is tokenized without special tokens. We prepend a [BOS] token to every training chunk and append an EOC token after every chunk, marking the boundary between consecutive chunks within a document. For the final chunk of each document (identified by the doc_end_flag), an <|endoftext|> token is additionally appended, signaling the document boundary. This structure preserves the document–chunk hierarchy in the token stream: the model can distinguish intra-document chunk boundaries within a document from boundaries between documents.

Chunk–concept alignment. The concept annotations produced by the annotator are chunk-level labels: each chunk is associated with a list of concept IDs, with no token-level labels. The [EOC] token serves as the delimiter that aligns concept annotations with their corresponding token spans in the training data. During training, the concept loss (Section 5) uses *OR-aggregation* across all tokens within a chunk to bridge this chunk-level supervision with the model's token-level concept activations. Figure 15 illustrates this structure.

4.6 Training data indexing for test-time attribution

The pipeline in Section 4.5 produces two artifacts: approximately 11 billion text chunks annotated with concepts by Stage 3 of the Atlas pipeline (Section 4.2.3), and, for each chunk, a 1024-dimensional embedding vector computed during the annotator's forward pass over the corpus. Because these embeddings are cached at annotation time, index construction reuses them directly and requires no additional annotator forward passes. These vectors form the index that supports test-time training data attribution.

Vector database. An exact (flat) index scores a query against every stored vector. While accurate, it imposes two costs that are each prohibitive at our scale: search latency that grows linearly in the number of vectors N, and memory that grows as $N \times d$ in full precision. We therefore use an approximate index—an Inverted File with Product Quantization (IVFPQ) index from the FAISS library (Johnson et al., 2019)—which addresses both costs: an inverted-file structure partitions the space so each query scans only a small fraction of the vectors, and product quantization compresses the stored vectors into compact codes.

With our configuration, product quantization reduces the per-vector payload from $4{,}096$ bytes (a full-precision 1024-dimensional vector) to 64 bytes, roughly a $64\times$ compression. We ℓ_2-normalize all vectors at build and search time, so inner-product retrieval is equivalent to cosine similarity. Even so, the pretraining data index occupies approximately 808 GB on disk, exceeding available RAM; we therefore store the inverted lists in a memory-mapped on-disk format, so only the lists touched by a query are paged into memory at search time, enabling search over the full corpus without loading the entire index into RAM.

Retrieval accuracy. Because IVFPQ is approximate, we assess how faithfully it reproduces exact search. On $1{,}000$ query chunks drawn from the index itself, we query the index using an $n_{\text{probe}} = 16$,

and measure recall@k as the fraction of queries whose own source chunk—the known ground-truth nearest neighbor—appears among the top-k retrieved results. The index achieves a **recall@10 of 96.8%**, indicating that the quantized search recovers the exact nearest neighbor in the large majority of cases while operating within the memory and latency budget imposed by a corpus of this scale.

5 Inherently interpretable architecture

In this section, we present the architectural and training choices to build a language model with inherently interpretable outputs, an instantiation of Section 3. We depart from the autoregressive paradigm: the masking-based training objective and attention structure of diffusion models are better suited to the interpretability properties we want to enforce (Section 5.1), and we adopt a causal block-attention pattern that retains efficient inference while preserving the diffusion training objective (Section 5.2). On top of this backbone, we introduce the concept module, a bottleneck inserted between the transformer and the language modeling head that routes every prediction through an explicit concept representation (Section 5.3). We then describe the training procedure that ties these components together (Section 5.4).

5.1 Beyond autoregressive models

Autoregressive (AR) models, built on a causal-attention transformer architecture, have been the standard choice for large language models, achieving impressive performance across a wide range of tasks (Achiam et al., 2023; Anthropic, 2024; Team et al., 2023; Liu et al., 2024; Grattafiori et al., 2024b; Bai et al., 2023). Beyond raw capability, the AR paradigm benefits from years of accumulated practical knowledge: training procedures, hyperparameter choices, and scaling laws have been studied extensively and documented in detail by the open-source community (OLMo et al., 2025; Grattafiori et al., 2024b; Team et al., 2024; Liu et al., 2024), making AR models a tempting default. However, their inductive biases do not align well with our objectives. We aim to build a model that supports faithful input attribution and concept-level control at inference time, and AR generation makes both properties harder to enforce. AR models predict one token at a time conditioned on a strictly causal context: concepts typically span multiple tokens rather than localizing to one, and attribution lacks a natural absence-of-information baseline. The same left-to-right factorization underlies well-documented failure modes, including the reversal curse (Berglund et al., 2024) and poor performance on tasks requiring graph backtracking (Ye et al., 2025).

The limitations above are not incidental to AR models; they follow from the left-to-right factorization itself, so addressing them calls for a different generative paradigm. Diffusion language models offer a more natural fit. The masking objective sidesteps each of the failures identified above and provides three properties we exploit throughout this work. First, it gives the model an explicit, trained representation of "no information at this position," the absence baseline that faithful input attribution requires and that strictly causal contexts cannot provide. Second, multiple tokens are predicted jointly at each denoising step, giving us a natural interface for concept-level control over phrases rather than individual tokens, directly addressing the fact that concepts span multiple tokens rather than localizing to one. Third, diffusion models generate tokens in any order, so the model can choose where in the sequence to express an intervened concept rather than being forced to commit at the next position. We describe our specific instantiation, Causal Diffusion, in the next section.

5.2 Causal Diffusion

Our objectives place two demands on the architecture. First, we want the model to operate over groups of tokens rather than single positions, since the concepts we interpret and steer (Section 5.3) typically span phrases rather than localizing to one token. Second, we want autoregressive-style inference efficiency: the KV caching and throughput that make AR models practical at scale. Masked diffusion gives us the first through its joint, any-order denoising, but, as we show below, its standard form sacrifices the second. The remainder of this section develops an attention structure that recovers both.

We build on masked diffusion models (MDMs; see Section 11 and Section 2). Standard MDMs use full bidirectional attention: every token attends to every other token in the sequence at each denoising step (Figure 16, panel b). Bidirectional context is what enables MDMs to denoise multiple tokens jointly in a single forward pass and to predict tokens in any order, since no position is privileged over another. The same property, however, prevents efficient inference. Because tokens that change between denoising steps are attended to by every other token, no representations can be cached across steps (Israel et al., 2026; Arriola et al., 2025). Each step recomputes the full attention over the full sequence, making inference substantially slower than autoregressive models of comparable size.

Block diffusion (Arriola et al., 2025) addresses this inference-time cost by modifying the attention pattern. The sequence is partitioned into blocks of fixed length b, and the attention mask is bidirectional within each block but causal across blocks (Figure 16, panel c). Generation proceeds one block at a time, and KV caches built from previously generated blocks can be reused across denoising steps. The training algorithm, however, requires concatenating a noisy and a clean copy of the sequence as input: noisy blocks supply the diffusion loss, while clean previous blocks supply context. This roughly doubles the per-step memory and FLOPs relative to standard MDM training.

We propose **Causal Diffusion**, which retains Block Diffusion's attention pattern but drops the clean copy of the sequence at training time (Figure 16, panel d). The training objective is the standard masked diffusion loss from Section 2.3, applied to a single sequence of tokens with a block-causal attention mask: bidirectional within each block, causal across blocks. Nie et al. (2025) showed that an MDM trained with full bidirectional attention can be sampled block-by-block at inference time with minimal quality degradation, suggesting that the masking objective does not require bidirectional context across the entire sequence. If block-causal attention is sufficient at inference, training under the same constraint costs nothing in expressive power: we obtain block-causal structure at half the training cost of Block Diffusion, without the inference-time bottleneck of standard MDMs. At inference, each new block is denoised by masked diffusion while conditioning on the keys and

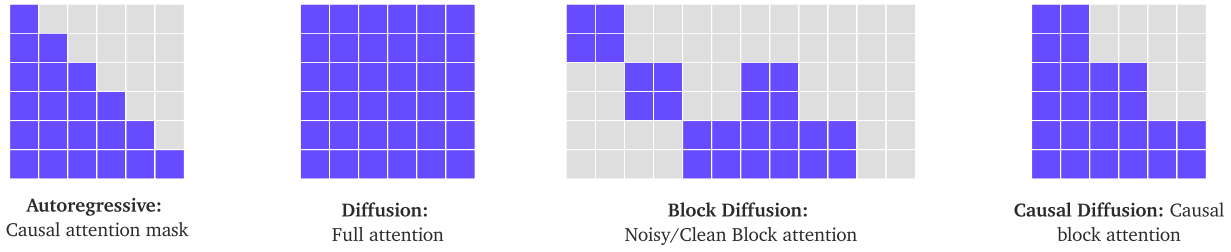

Figure 16. Attention patterns for Autoregressive, Diffusion, Block Diffusion, and Causal Diffusion models.

values cached from all previous blocks; once generated, its own keys and values are appended to the cache, exactly as in autoregressive decoding. The result is an MDM that retains diffusion's par-

allelism and any-order flexibility within each block while inheriting autoregressive-style KV caching across blocks.

5.3 Concept module

One way to build an inherently interpretable language model is to decompose the hidden representation into a set of concepts and use those concepts, combined through a simple and interpretable function, for the model's predictions. Two properties follow directly from this decomposition: faithful concept attribution, since each concept's contribution to a given output can be computed directly, and concept steering, since we know how each concept is represented and can therefore bias the model towards or away from it. We achieve this with an additive bottleneck: the hidden representation is reconstructed as a sum of concept contributions before being passed to the language modeling head. We refer to this as our *concept module*.

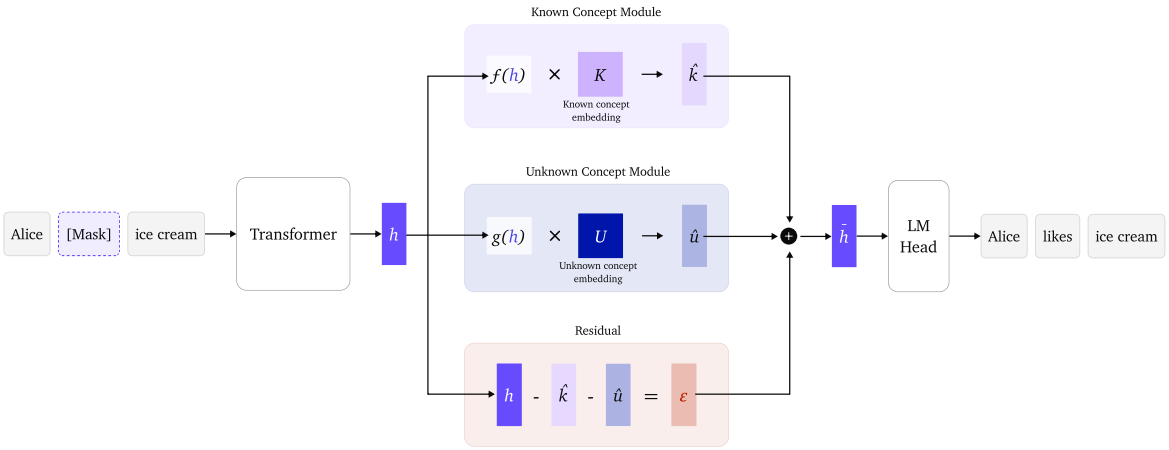

Figure 17. The concept module bottleneck. The transformer hidden state h is decomposed into known concept contributions $\hat{k}$, unknown concept contributions $\hat{u}$, and a residual ε.

The concept module sits between the transformer backbone and the language modeling head (Figure 17). For each input token, the transformer produces a hidden state $h \in \mathbb{R}^d$, which is normally passed straight to the language modeling head. The concept module instead decomposes h into three additive components:

$$\bar{h} = \hat{k} + \hat{u} + \varepsilon, \tag{5}$$

where $\hat{k}$ is a weighted sum of *known* concept embeddings, $\hat{u}$ is a weighted sum of *unknown* concept embeddings, and $\varepsilon = h - \hat{k} - \hat{u}$ is the residual term. Only $\bar{h}$ is passed to the LM head, so every output logit becomes a linear function of concept activations. We apply dropout with rate p_ε to ε during training, to discourage the model from relying on the residual channel for prediction.

Both $\hat{k}$ and $\hat{u}$ are computed from h via two small heads. The known head f produces concept activation probabilities for the labeled concept set, and the unknown head g produces activation probabilities for a set of unknown concepts:

$$k = \sigma(f(h)) \in \mathbb{R}^n, \qquad u = \sigma(g(h)) \in \mathbb{R}^m, \tag{6}$$

where f and g are small learnable networks, σ denotes the elementwise sigmoid, n is the number of known concepts, and $m \gg n$ is the number of unknown concepts. Each concept i has a

learned embedding $K_i \in \mathbb{R}^d$ (or U_j for unknown), analogous to a token embedding, and the concept-weighted hidden states are

$$\hat{k} = \sum_{i=1}^{n} k_i K_i, \qquad \hat{u} = \sum_{j=1}^{m} u_j U_j. \tag{7}$$

The full set of concept embeddings forms a vocabulary of concepts, in the same way that token embeddings form a vocabulary of tokens. Given the large number of unknown concepts, storing the unknown embedding matrix $U \in \mathbb{R}^{m \times d}$ directly would dominate the parameter count. We therefore factorize it as a low-rank product $U = AB$, with $A \in \mathbb{R}^{m \times r}$ and $B \in \mathbb{R}^{r \times d}$ for rank $r \ll d$, reducing the parameter count from md to $r(m + d)$ while preserving capacity.

At inference time, the additive structure of $\bar{h}$ makes every prediction a transparent function of the concept activations. Because the LM head is linear, the logit for any output token decomposes as

$$\ell_y = \hat{k}^\top W_y + \hat{u}^\top W_y + \varepsilon^\top W_y, \tag{8}$$

where W_y is the row of the LM head corresponding to token y. Each term is exact: there is no approximation involved in attributing a prediction to known concepts, unknown concepts, or the residual. This decomposition is what allows the model to support faithful concept attribution.

Taken together, these components make the concept module a self-contained, modular interface to the model's predictions: known and unknown concepts, their embeddings, and the residual are separate, inspectable channels, and every output is an explicit, additive function of them. This organization is what makes the attribution and steering described in Section 6 possible.

5.4 Model training

The architecture described in Section 5.3 forces every prediction to pass through an additive concept representation. In effect, it builds in the linear representation hypothesis (Park et al., 2023), the idea that high-level concepts are encoded as linear directions in representation space. Rather than hoping this property emerges, as it may or may not in a standard language model, our architecture enforces it by construction: every output is an explicit linear function of concept activations. This structural guarantee, however, does not by itself ensure that the concept module learns useful concepts, that the language modeling head produces fluent text, or that the known and unknown concepts encode complementary information. Each of these properties has to be optimized for. We describe the loss objectives that target them in Section 5.4.1, and the training dynamics that control how those losses are applied over time in Section 5.4.2.

5.4.1 Loss objectives

Language modeling loss. The primary objective is the masked diffusion loss $\mathcal{L}_{\text{MDM}}$ from Section 2.3, applied to the bottlenecked hidden state $\bar{h}$ rather than the raw transformer output h. Every token prediction is therefore evaluated as a function of the concept module's output. We denote this loss $\mathcal{L}_{\text{LM}}$.

Concept loss. The ground-truth concept labels are produced at the chunk level by the annotation pipeline of Section 4. A chunk is a contiguous span of tokens terminated by an `[EOC]` token (Figure 15). The labels are positive-only: a label tells us a concept appears somewhere in a chunk, but not at which token. We accommodate this weaker supervision signal with an OR-aggregation across

the chunk. For known concept c, let $k_{c,t}$ denote its predicted activation at token t (the c-th entry of k from Equation (6)). The probability that c appears at least once in the chunk is

$$k_c^{\text{chunk}} = 1 - \prod_{t \in \text{chunk}} (1 - k_{c,t}). \tag{9}$$

Let $y_c \in \{0, 1\}$ denote the ground-truth chunk-level label for concept c. The concept loss is the binary cross-entropy between the aggregated probability and the chunk label, summed over all known concepts:

$$\mathcal{L}_{\text{concept}} = -\sum_{c=1}^{n} \Big[y_c \log k_c^{\text{chunk}} + (1 - y_c) \log(1 - k_c^{\text{chunk}}) \Big]. \tag{10}$$

The OR-aggregation is satisfied as soon as the concept is predicted at any one token in the chunk, consistent with our chunk-level supervision.

Reconstruction loss. The unknown head is trained to represent the part of the hidden state that is not captured by the known concepts. Given the ground-truth labels for known concepts, the ideal known representation and the corresponding target for the unknown head are

$$\hat{k}^{\text{GT}} = \sum_{i=1}^{n} k_i^{\text{GT}} K_i, \qquad \hat{u}^{\text{GT}} = h - \hat{k}^{\text{GT}}, \tag{11}$$

where $\hat{u}^{\text{GT}}$ is the residual that remains after subtracting $\hat{k}^{\text{GT}}$ from the transformer hidden state. The unknown head is trained to match this target under a mean-squared error, averaged over masked positions in the minibatch:

$$\mathcal{L}_{\text{rec}} = \frac{1}{|\mathcal{M}|} \sum_{t \in \mathcal{M}} \|\hat{u}_t - \hat{u}_t^{\text{GT}}\|_2^2, \tag{12}$$

where $\mathcal{M}$ is the set of masked token positions in the minibatch. When $\hat{u} = \hat{u}^{\text{GT}}$, the residual term in the bottleneck satisfies $\varepsilon = 0$.

Independence loss. The reconstruction loss alone does not prevent the unknown head from encoding information that is already represented by the known concepts. To discourage such redundancy, we penalize the statistical dependence between the known and unknown representations using a normalized cross-covariance penalty in the spirit of the Hilbert-Schmidt Independence Criterion with a linear kernel (Mooij et al., 2009; Greenfeld and Shalit, 2020), following its use for concept decoupling in Andersson et al. (2026). Over a minibatch of B token representations, let

$$\Phi = H_k - \mathbf{1}\boldsymbol{\mu}_{\hat{k}}^\top, \qquad \Psi = H_u - \mathbf{1}\boldsymbol{\mu}_{\hat{u}}^\top, \tag{13}$$

where $H_k, H_u \in \mathbb{R}^{B \times d}$ stack the per-token $\hat{k}, \hat{u}$ components, $\boldsymbol{\mu}_{\hat{k}}, \boldsymbol{\mu}_{\hat{u}} \in \mathbb{R}^d$ are their column means, $\mathbf{1} \in \mathbb{R}^B$ denotes the all-ones vector, and $\|\cdot\|_F$ is the Frobenius norm. The independence loss is

$$\mathcal{L}_{\text{indep}} = \frac{1}{d^2(B-1)} \left\|\Psi^\top \Phi\right\|_F^2. \tag{14}$$

In practice, gradients flow only through the unknown representation Ψ; the known representation Φ is treated as a fixed input. Minimizing $\mathcal{L}_{\text{indep}}$ drives the cross-covariance between $\hat{k}$ and $\hat{u}$ toward zero, encouraging the two heads to encode complementary rather than redundant information of the hidden state.

Final training objective. The four losses are combined linearly with non-negative weights:

$$\mathcal{L} = \mathcal{L}_{\text{LM}} + \lambda_{\text{concept}}\mathcal{L}_{\text{concept}} + \lambda_{\text{rec}}\mathcal{L}_{\text{rec}} + \lambda_{\text{indep}}\mathcal{L}_{\text{indep}}. \tag{15}$$

The backbone and concept module are optimized jointly under this objective, where $\lambda_{\text{concept}}, \lambda_{\text{rec}}, \lambda_{\text{indep}}$ are hyperparameters. For both auxiliary losses $\mathcal{L}_{\text{rec}}$ and $\mathcal{L}_{\text{indep}}$, gradients are detached so that only the unknown head is updated.

5.4.2 Training dynamics

Per-block masking schedule. Standard masked diffusion models apply a single noise level $t \sim \mathcal{U}(0, 1)$ to the entire sequence at each training step. In the block-causal architecture (Section 5.2), we treat each block as an independent unit and sample a separate noise level $t_b \sim \mathcal{U}(0, 1)$ for each block b, so two blocks within the same context window can be at different stages of the denoising process simultaneously (Figure 18). The model thus observes a richer distribution of partially-denoised contexts at every training step than under a single global noise level.

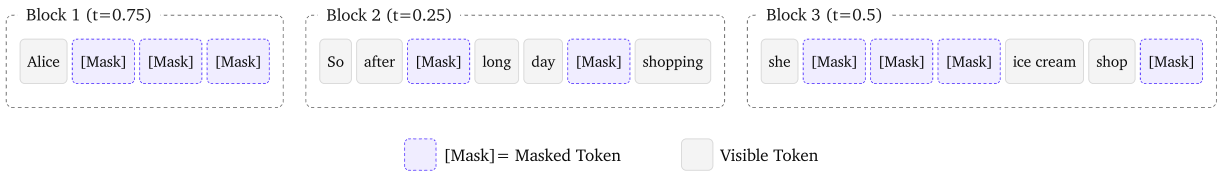

Figure 18. Per-block masking schedule. Each block samples an independent noise level $t_b \sim \mathcal{U}(0, 1)$, so blocks in the same sequence are denoised to different degrees within a single training step.

Concept teacher forcing schedule. The known head's predictions are unreliable early in training, and even once they stabilize, routing $\bar{h}$ through the predicted activations $\hat{k}$ allows the language modeling loss to push those activations to encode information beyond the labeled concepts, a failure mode known as concept leakage (Mahinpei et al., 2021). Koh et al. (2020) address this by training the concept head independently of the downstream model and feeding ground-truth concepts forward at every step. We adopt the same substitution idea but apply it on a schedule: with probability $\alpha_{\text{known}}(s)$ at training step s, we replace the predicted known representation $\hat{k}$ with its ground-truth analogue $\hat{k}^{\text{GT}} = \sum_i k_i^{\text{GT}} K_i$ from Equation (11) when forming $\bar{h}$, which we refer to as teacher forcing. We anneal α_{known} from 1 at the start of training (full teacher forcing) to a smaller steady-state value, so the model relies progressively on its own predictions as the head becomes more accurate.

Unknown concept teacher forcing schedule. The unknown head faces the same early-training instability as the known head, but no ground-truth labels exist to substitute for its predicted activations. Instead, we substitute the ideal target itself: from Equation (11), the unknown embedding should reconstruct $\hat{u}^{\text{GT}} = h - \hat{k}^{\text{GT}}$, which we can compute directly from the transformer hidden state and the labeled concepts. With probability $\alpha_{\text{unknown}}(s)$ at training step s, we replace the predicted $\hat{u}$ with $\hat{u}^{\text{GT}}$ when forming $\bar{h}$. As with α_{known}, we anneal α_{unknown} from 1 to a smaller steady-state value, so the language modeling head is shielded from a poorly-trained unknown head early on and learns to rely on the predicted unknown embedding as it becomes accurate.

A consolidated reference for the symbols introduced in this section is provided in Appendix A.

6 Interpretability capabilities

The architecture of Section 5 produces models that are both *interpretable* and *controllable*. Interpretability answers "why a prediction was produced.": through attribution, we identify the inputs, concepts, and training data associated with a prediction. Controllability is enabled through steering, which modifies the concept representation at inference time to control how selected concepts affect the model's behavior. We describe attribution and steering in the following sections.

6.1 Attribution

Attribution asks why Steerling produced a given output. We approach it from three angles:

- Input attribution identifies influential input tokens (Section 6.1.1).
- Concept attribution traces influential internal concepts (Section 6.1.2).
- Training data attribution retrieves similar training examples (Section 6.1.3).

Figure 19 shows the three methods applied to a single output chunk.

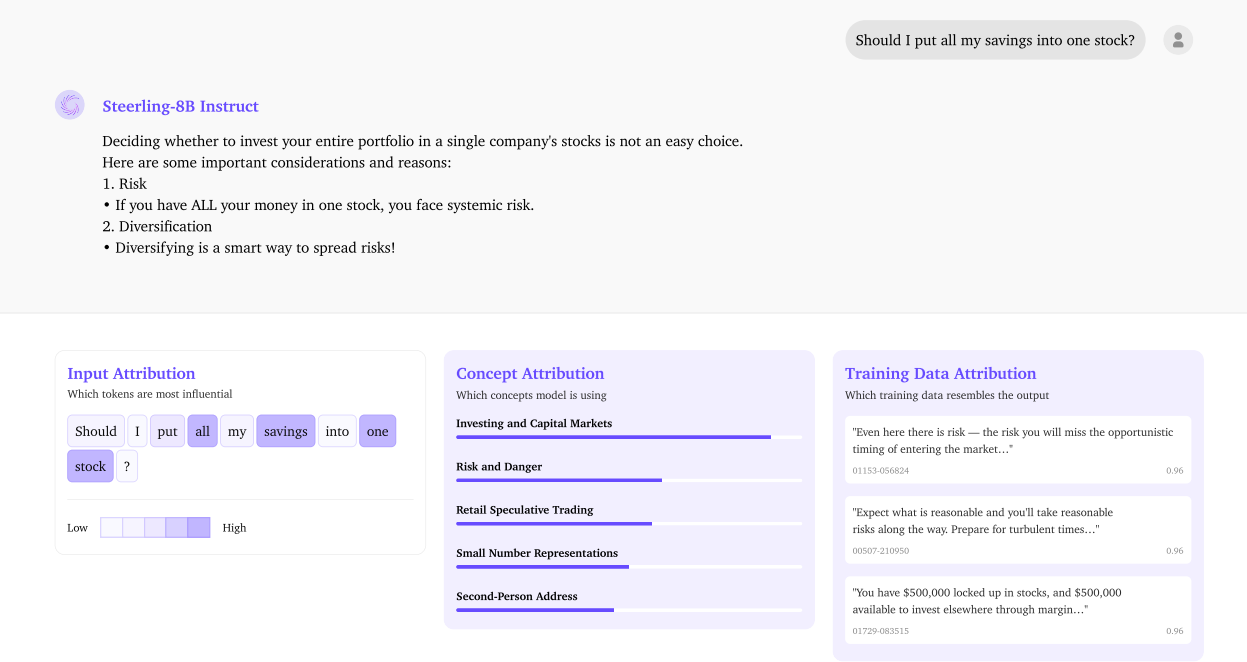

Figure 19. Three views of attribution for one output chunk. **(A)** Per-token Integrated Gradients scores over the prompt. Tokens that contribute most to the chunk are shaded more intensely. **(B)** Top contributing concepts, ranked by their chunk-level contribution. **(C)** Retrieved training chunks, ranked by similarity to the chunk's representation. We can inspect their source URLs for verification.

6.1.1 Input attribution

Input attribution answers: *which input tokens most influenced a given output?* We use Integrated Gradients (Equation 16) with the `[MASK]` embedding as the baseline. The baseline determines what the attribution *measures*: how the output changes as the input moves from the baseline to its actual

value. [MASK] makes that comparison meaningful for our model. The diffusion objective trains the model to predict masked tokens at every position, so [MASK] becomes a learned representation of "no information at this position," and the integration runs from this learned "absent" state to the actual token along a path of states the model was trained on. Figure 19A shows the per-token scores for the highlighted chunk in this example. Autoregressive models have no such trained baseline: zero embeddings and padding tokens are out-of-distribution states the model never learned to read as an absence of information.

We compute attribution using Integrated Gradients (Sundararajan et al., 2017a) on the token embeddings. For an input token x^i with embedding T_{x^i}, we integrate the gradient of the output logit ℓ_y along the straight-line path from the baseline embedding $T_{\texttt{[MASK]}}$ to T_{x^i}, giving a score

$$\phi(x^i, y) = \frac{1}{S} \sum_{s=1}^{S} \left(T_{x^i} - T_{\texttt{[MASK]}}\right)^\top \nabla_{T^{(s)}} \ell_y, \tag{16}$$

where $T^{(s)} = T_{\texttt{[MASK]}} + \frac{s}{S}(T_{x^i} - T_{\texttt{[MASK]}})$ is the s-th of S interpolation points along that path. Replacing a token with [MASK] is an operation the model has performed countless times during training, so the Integrated Gradients path traces a well-defined direction through the model's learned representation space.

6.1.2 Concept attribution

Concept attribution answers: *which internal concepts most influenced a given output?* The additive bottleneck of the concept module already decomposes the logit of any output token exactly into known-concept, unknown-concept, and residual terms (Equation 8). Figure 19B ranks the top contributing concepts for the highlighted chunk.

We report attribution at the chunk level, consistent with how concepts are supervised (Section 5.4). The contribution of a concept to a chunk is the sum of its per-token contributions over the tokens in that chunk:

$$\Gamma_i^{\text{known}} = \sum_{t \in \text{chunk}} k_{i,t}\, K_i^\top W_{y_t}, \qquad \Gamma_j^{\text{unknown}} = \sum_{t \in \text{chunk}} u_{j,t}\, U_j^\top W_{y_t}, \tag{17}$$

where $k_{i,t}$ and $u_{j,t}$ are the activations of known concept i and unknown concept j at the token y_t produced at position t, and an analogous sum gives the residual's contribution. Ranking concepts by these chunk-level contributions identifies which concepts drove the output, while the residual captures the part of the output that the concept inventory does not explain.

6.1.3 Training data attribution

Training data attribution (TDA) answers: *which training examples are most influential to a given output?* It enables applications such as alignment fine-tuning, factual provenance tracing, and auditing whether a model generalizes from proprietary fine-tuning data or falls back on pretraining knowledge. Unlike concept attribution, data attribution does not derive its scores from the model's own computation. We do not estimate the causal effect of removing a training example, as classical influence-function methods do (Koh and Liang, 2017); at our scale the required Hessian inverse is intractable. Instead we frame attribution as *approximate semantic-similarity retrieval*: given an output, we retrieve the training chunks whose representations are most similar to it. This is a scalable proxy for influence rather than a faithful or causal account. The retrieval view suits Steerling in particular: because its representations are aligned to human-interpretable concepts (Section 5),

semantically related chunks lie close together in representation space, so nearest-neighbor search tends to surface training chunks that share concepts with the output.

Because data attribution retrieves rather than computes, its central difficulty is representational. The query vector we extract from Steerling lives in the model's internal representation space, whereas the index stores corpus embeddings produced by a different model, so an output's internal representation cannot be matched against the index directly. We bridge them in two steps: forming a query representation from an output, then mapping it into the index's space. Given a model output, we segment it into chunks at the `[EOC]` token and forward each chunk through the language model, mean-pooling over its token positions to obtain a single vector that combines the known-head, unknown-head, and residual components of the hidden state. A learned *projection*—a small MLP trained with a cosine-similarity objective—maps this internal vector into the corpus embedding space of the index built in Section 4.6, placing an output and its related training chunks close together. The projected query is then matched against the index by approximate nearest-neighbor search, and the nearest chunks are returned as the attributed sources. Figure 19C shows the top-ranked retrieved chunks for the highlighted output chunk, ranked by cosine similarity; their source URLs can be inspected for verification.

6.2 Steering

Steering is the ability to control model output without using prompts. This is enabled by the concept module and is not possible in other existing models. We steer generation at inference time, without updating any weights, by acting directly on the concept representations the model already uses. Steering can push generation toward a target concept (amplification) or away from it (suppression), and can be applied to both the known concepts supervised during training and the unknown concepts the model discovers on its own.

6.2.1 Steering operation

Our model learns a direction (or embedding) for each concept and uses it internally to form predictions. We can steer the model toward a given concept by injecting the direction the model has already learned for it into the hidden representation. Injecting a direction into the hidden states to steer generation is a well-established technique (Subramani et al., 2022; Turner et al., 2023; Rimsky et al., 2024; Zou et al., 2023), but these methods must first extract the direction post-hoc from a trained model's activations, so it is only an estimate of how a concept is represented and varies with the procedure used to recover it. Our architecture is better suited to steering because the direction is not estimated: each concept's embedding K_c is a model parameter the model itself uses, so we inject a direction the model already relies on rather than one fit after the fact.

Concretely, we take the concept embedding K_c and normalize it to unit length, giving a steering direction $e_c = K_c/\|K_c\|_2$. To steer toward several concepts at once, we sum their embeddings then normalize. We then add this direction, scaled by a steering strength γ, to the transformer hidden state at every masked position, at every layer from L_{inj} onward. The steering signal accumulates as the representation propagates toward the bottleneck:

$$h_t^{(l)} \leftarrow h_t^{(l)} + \gamma\, e_c, \qquad l \geq L_{\text{inj}}. \tag{18}$$

Because each concept aligns with the LM head differently, the effect of a given γ is not comparable across concepts: the same strength shifts the output logits more for some concepts than others. We

therefore calibrate γ per concept so that its largest effect on any output token equals a fixed target τ:

$$\gamma = \frac{\tau}{\text{peak}(e_c)}, \qquad \text{peak}(e_c) = \max_{y \in V} e_c^\top W_y, \tag{19}$$

where $\text{peak}(e_c)$ is the largest logit shift the direction e_c can induce over the vocabulary V. A single global τ then gives adaptive steering strength across concepts without per-concept tuning.

6.2.2 Steering direction

The sign of the steering strength γ sets the direction of the intervention. Amplification pushes generation toward the target concept, while suppression pushes it away.

Amplification. Amplification takes $\gamma > 0$, so the injection of Equation (18) moves the hidden state along e_c, toward the concept c. The concept direction is added before the bottleneck, propagating the boosted activation through the concept module then the LM head to increase the logits of concept-expressing tokens.

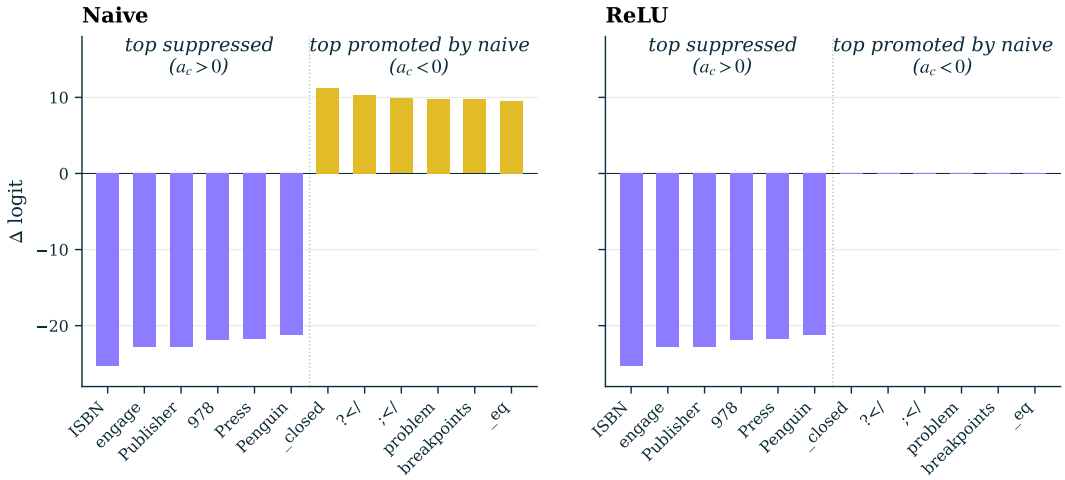

Figure 20. ReLU-gated concept suppression avoids unintended token promotion during negative steering. Naive subtraction suppresses tokens positively aligned with the target concept, but promotes tokens with negative alignment. For an *academic and commercial publishing* concept, this boosts unrelated tokens such as *problem* and *breakpoints*. The ReLU-gated update preserves suppression of publishing-related tokens while leaving anti-aligned tokens unchanged.

Suppression. Suppression eliminates text associated with a target concept c. We combine two mechanisms to ensure the concept is properly suppressed: a negative hidden-state injection (Equation (18) with $\gamma < 0$) that pushes the transformer representation away from c, and a ReLU-gated logit mask that subtracts c's contribution from the LM head output. We explain why we use the ReLU gate below.

Since the LM head is a linear projection on top of the sum of concept embeddings, the contribution of concept c to the output is its alignment with each vocabulary token:

$$a_c = W e_c \in \mathbb{R}^{|V|}, \qquad a_c[v] = W_v^\top e_c, \tag{20}$$

where $a_c[v]$ is the concept's contribution to the logit of token v. Subtracting this contribution directly ($\ell_v \to \ell_v - s\, a_c[v]$ with suppression strength $s > 0$) will suppress tokens with $a_c[v] > 0$ as intended,

but simultaneously *promotes* tokens with $a_c[v] < 0$, making unrelated tokens that happen to be anti-aligned with the concept (Figure 20) dominate generation. We gate the subtraction with a ReLU so that only positively aligned tokens are affected:

$$\ell_v \longrightarrow \ell_v - s \cdot \mathrm{ReLU}\big(a_c[v]\big). \tag{21}$$

This suppresses the tokens aligned with the concept while leaving anti-aligned tokens untouched.

7 Interpretability Metrics

This work is built on the claim that interpretability is best treated as a design constraint rather than a post-hoc analysis problem. If that claim is correct, then a model trained with interpretability built into its architecture and objectives should be more interpretable than one trained without, not in some abstract sense, but in ways one can measure directly. This section makes that claim measurable. The metrics defined below test the properties the concept module of Section 5.3 was designed to deliver.

The concept module was trained to detect a set of concepts in context, via the concept loss. The model was trained to route its predictions through those concepts rather than through the residual, via the reconstruction loss. The known and unknown concepts were trained to be disentangled, so they do not encode the same information, via the independence loss. Each of these properties has a corresponding metric, and each metric tests a necessary condition for the architecture to be working as built. A fourth metric asks whether the learned concept embeddings point at semantically related tokens; this property is not directly supervised by any training objective, and is included to test whether the architecture produces token-level interpretability beyond what was directly optimized for.

These metrics are familiar from the broader interpretability literature, where post-hoc methods such as sparse autoencoders, probes, and steering vectors are evaluated along similar axes of concept detection and disentanglement (Wu et al., 2025; Bhalla et al., 2024). These post-hoc methods exist because representations in standard transformers are in superposition (Elhage et al., 2022; Bricken et al., 2023a): more learned directions than dimensions, packed into overlapping subspaces that have to be recovered after training. The concept module sidesteps superposition by construction: each concept has its own learned embedding, so the metrics do not have to ask whether a recovered direction behaves like a concept, but whether the concepts the model was trained on are doing the work the architecture assigned to them.

Metric	Direction	What it measures
Concept Loss	↓	Does the concept module detect the right concepts?
Concept Independence Loss	↓	Are known and unknown heads disentangled?
Concept Contribution	↑	Do predictions route through the concept module?
Known Concept Alignment	↑	Do concept embeddings point at related tokens?

Table 5. Interpretability metrics. ↑ or ↓ indicates the direction of better performance.

Concept Loss. The concept loss (Equation 10) measures whether the concept module assigns the right concepts to a given chunk. The metric is the OR-aggregated binary cross-entropy between predicted concept presence (Equation 9) and ground-truth chunk-level labels, computed over a held-out validation set of chunks annotated by the pipeline of Section 4. A low value indicates that the

concept module identifies concepts in unseen text in the same way it learned to identify them during training.

Concept Independence Loss. The independence loss (Equation 14) measures the linear statistical dependence between the known and unknown concept representations. The metric is computed over per-token $\hat{k}$ and $\hat{u}$ on a held-out validation set. A low value indicates that the two representations carry linearly independent information. A linear kernel was chosen because the bottleneck composes its components additively, $\bar{h} = \hat{k} + \hat{u} + \varepsilon$, and the LM head is linear, so every output logit decomposes linearly into contributions from each pathway (Equation 8). Linear independence between $\hat{k}$ and $\hat{u}$ is therefore the exact property concept attribution requires: if the two representations are linearly independent in expectation, their contributions to any logit are additively separable, and the known and unknown pathways do not encode overlapping information that would inflate one attribution at the expense of the other.

Concept Contribution. The concept contribution metric measures how much of each output prediction is explained by the concept module rather than by the residual. From the logit decomposition in Equation (8), the contribution of the concept module to the logit of any token y is the sum of the known and unknown terms; the residual term $\varepsilon^\top W_y$ denotes what has not been captured by the concept module. The metric is the fraction of total logit magnitude attributable to the concept module:

$$\text{Concept Contribution} = \frac{|\hat{k}^\top W_y| + |\hat{u}^\top W_y|}{|\hat{k}^\top W_y| + |\hat{u}^\top W_y| + |\varepsilon^\top W_y|}, \tag{22}$$

averaged over predictions on a held-out validation set. A high value indicates that predictions can be mostly attributed to the concept module. The reconstruction loss (Equation 12) creates training pressure on this property by driving $\varepsilon \to 0$, but the metric is computed at the logit level rather than on $\|\varepsilon\|$ directly, so it tests the property that matters for attribution faithfulness: how much of the prediction are the concepts responsible for.

Known Concept Alignment. Each known concept has a learned embedding $K_c \in \mathbb{R}^d$ that the LM head projects into vocabulary space, so the top-k tokens for concept c are $T_k(c) = \text{TopK}(WK_c)$. The known concept alignment metric asks whether these top tokens semantically match the human-assigned label and description that the concept was originally annotated with by the pipeline of Section 4. An LLM-judge is given the concept's label, its description, and the top tokens, and rates the alignment on a 1-5 scale, with 5 indicating that the top tokens unambiguously represent the named concept and 1 indicating no relationship. Details on the judge and the full prompt are given in Appendix E. The metric is the mean rating over a randomly sampled subset of concepts from the library.

Known concept alignment is not directly supervised by the training objective. The concept loss (Equation 10) supervises chunk-level concept presence using positive-only labels; it does not connect a concept embedding K_c to the specific tokens that lexically express concept c. The language modelling loss supervises next-token prediction; it does not connect token predictions to the concept vocabulary. Any alignment between K_c and semantically-related tokens therefore arises from the joint optimization of these two objectives, not from a direct training signal.

8 Scaling laws

A long-standing concern in the interpretability literature is that constraining a model's representations to be human-understandable necessarily costs capability (Rudin, 2019; Doshi-Velez and Kim, 2017; Koh et al., 2020). A second concern, less studied, is that even when interpretability properties hold at small scale, there is no guarantee they survive at the scales where models actually get deployed (Wei et al., 2022a). This section tests both concerns empirically. Two model families, autoregressive (AR) and causal diffusion (CDLM), together with their interpretable counterparts, are trained across three orders of magnitude of compute. Scaling laws are fit to the resulting checkpoints for capability (compute-optimal loss) and for the interpretability metrics defined in Section 7, and each fit is then used to extrapolate to Steerling, a CDLM+Concept model trained at 8B parameters and 1.35T tokens. Specifically, this section addresses two research questions:

- **RQ1.** Do inherently interpretable architectures preserve compute-optimal scaling?
- **RQ2.** Do the interpretability properties scale favorably with compute?

Across model families and three orders of magnitude of compute, adding the concept module to either backbone shifts the compute-optimal scaling exponents by **a small, fixed per-backbone offset**. The interpretability metrics defined in Section 7 **improve favorably with compute on both backbones**, with all four metrics scaling in the expected direction. Extrapolating the small-scale fits to Steerling **predicts the deployed model's validation loss within** 0.11 **nats** and bounds its interpretability metrics within their natural ranges. A consolidated reference for the symbols introduced in this section is provided in Appendix F.

8.1 Setup

Model families. Four model families are compared. **AR** is a standard autoregressive transformer; **CDLM** is the causal block-diffusion model defined in Section 5.2; and **AR+Concept** and **CDLM+Concept** are their inherently interpretable counterparts, obtained by inserting the concept module of Section 5.3 between the transformer hidden state and the LM head. The two backbones are held fixed across all experiments; only the presence of the concept module changes. The concept library used by the +Concept families is the one constructed in Section 4, fixed across all model sizes.

IsoFLOP sweep. Each family is trained across four IsoFLOP slices, with slice targets reported in Table 24. Each slice contains four to six model sizes whose token counts are adjusted so that their total compute lands within $\pm 15\%$ of the slice target. The slice targets differ between base and +Concept families: at the smallest backbones, the concept module's parameter overhead exceeds the backbone itself, making compute-optimal estimates unreliable in the lowest IsoFLOP slice. Slightly higher targets are therefore used for the +Concept families. Full architecture configurations and slice targets are reported in Appendix G.

Training. All runs share the same data, optimizer, and schedule. Pretraining is on the real-data subset of Nemotron-CC-HQ (Su et al., 2025) at sequence length $N = 4096$. Optimization uses AdamW under a warmup-stable-decay (WSD) schedule (Hu et al., 2024): a short linear warmup, an extended stable phase at peak learning rate, and a final 20% linear decay to zero. Optimizer hyperparameters, tokenizer, and per-family configurations are reported in Appendix G.

Each IsoFLOP checkpoint is annealed independently rather than sharing a single annealing tail. The 80% stable-phase trajectory is treated as a reservoir of starting points; a 20% linear decay is run from each one to its final loss. This is more expensive than estimating annealed losses from non-annealed checkpoints (von Rütte et al., 2025), but it yields a dense compute-loss sweep without a separate full run per slice. The cheaper estimation procedure produced inconsistent results across model families and sizes (Appendix I).

Parameters and FLOPs calculation. Following Bi et al. (2024), per-token forward+backward FLOPs for the AR and CDLM backbones are

$$M_{\text{base}} = 6P + 12LdN, \tag{23}$$

where P is the non-embedding parameter count, L the number of layers, d the hidden dimension, and N the sequence length. Total training FLOPs are $C = M \cdot D$ for D training tokens.

The +Concept families add two heads acting on every token (Section 5.3). The known head scores all n concepts through a predictor of size $d \times n$, then composes the result through a separate embedding matrix of the same size via top-k_{known} selection. The two matmuls give a per-token cost of $2dn$. The unknown head is factorized through low-rank embeddings: a down-projection $d \to R$, a predictor over m unknown concepts, top-k_{unknown} selection, and composition through factorized embeddings, for a per-token cost of $2dR + Rm$. Combined:

$$M_{\text{+Concept}} = \underbrace{6P + 12LdN}_{\text{backbone}} + \underbrace{6\big(2dn\big)}_{\text{known head}} + \underbrace{6\big(2dR + Rm\big)}_{\text{unknown head}}. \tag{24}$$

Concept module hyperparameters and per-backbone parameter counts are reported in Appendix G.

The parameter cost of the concept module shrinks rapidly with scale. For a fixed concept library, the concept module adds a parameter overhead of $O(d)$, while backbone parameters scale as $O(d^2 L)$. The relative overhead therefore decays rapidly: the concept module accounts for $\sim 89\%$ of the total parameter count at 10M, $\sim 4\%$ at Steerling, and under 1% at frontier scales (Figure 21).

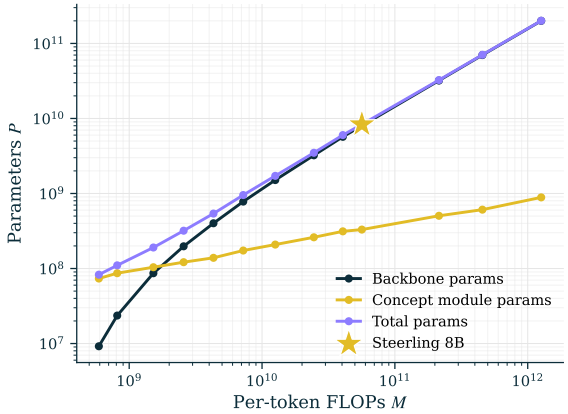

Figure 21. Concept module overhead as a fraction of total parameters, vs. backbone size.

8.2 Compute-Optimal scaling

Compute-optimal scaling is fit per family on the checkpoints from Section 8.1, yielding a set of $(P_i, D_i, \mathcal{L}_i)$ triples per family giving parameter count, training tokens, and validation loss, organized

into four IsoFLOP slices. The exponents are then compared pair by pair across each backbone's interpretable and non-interpretable variant, allowing a direct test of **RQ1: do inherently interpretable architectures preserve compute-optimal scaling?**

8.2.1 Methodology

Validation losses. For autoregressive models (AR, AR+Concept), $\mathcal{L}_i$ is validation cross-entropy on a held-out subset of the Nemotron pretraining corpus. For diffusion models (CDLM, CDLM+Concept), $\mathcal{L}_i$ is a Monte Carlo estimate of the masked diffusion model (MDLM) ELBO bound on the per-token negative log-likelihood (von Rütte et al., 2025; Sahoo et al., 2026; Nie et al., 2024). Per batch, a single noise level $t \sim \mathcal{U}(10^{-3}, 1-10^{-3})$ is sampled and shared across the batch, each token position is independently masked with probability t, a forward pass is run, and the mean cross-entropy on the masked positions is computed. The validation loss is the average over batches. Note that this estimator integrates over $t \in (0,1)$, while training samples $t \sim \mathcal{U}(0.05, 0.95)$; the two are therefore unbiased estimators of the same MDLM objective on slightly different intervals and will not match numerically. Details on the ELBO estimator and Monte Carlo sampling are given in Appendix H.

Step 1: Per-slice parabola in $\log P$. Within each IsoFLOP slice (fixed target C), the per-size $(P_i, \mathcal{L}_i)$ pairs are fit to a parabola in $\log P$ (Hoffmann et al., 2022):

$$\mathcal{L}(\log P;\, C) = a(C)\left(\log P - \log P^*(C)\right)^2 + \mathcal{L}^*(C), \tag{25}$$

where $a(C)$, $\log P^*(C)$, and $\mathcal{L}^*(C)$ are free parameters, fit by nonlinear least squares (Levenberg-Marquardt). The fitted $\log P^*(C)$ locates the parameter count that minimizes loss at compute budget C, and $\mathcal{L}^*(C)$ is the corresponding loss. Doing this once per slice produces four $(C_i, P_i^*, \mathcal{L}_i^*)$ triples per family.

Step 2: Power laws across compute budgets. With the four per-slice triples, we fit two power laws,

$$P^*(C) = a_P\, C^{\alpha_P}, \qquad \mathcal{L}^*(C) = a_L\, C^{\alpha_L}, \tag{26}$$

each by ordinary least squares on log-transformed data: α_P and $\log a_P$ are the slope and intercept of $\log P^*$ regressed on $\log C$, and analogously for α_L, $\log a_L$. The exponents α_P and α_L characterize how the compute-optimal parameter count and loss scale with available compute; the compute-optimal token exponent α_D is estimated from the joint fit of Step 3.

Step 3: Joint parametric fit for the irreducible loss. Steps 1 and 2 yield α_P and α_L but no irreducible-loss term. Following Hoffmann et al. (2022); Ni et al. (2025), the joint loss surface

$$\mathcal{L}(P, D) = \mathcal{L}_\infty + \frac{A_P}{P^\alpha} + \frac{A_D}{D^\beta} \tag{27}$$

is fit on all $(P_i, D_i, \mathcal{L}_i)$ observations pooled across the four IsoFLOP slices, not just the slice minima. The five free parameters are $\mathcal{L}_\infty$, A_P, A_D, α, β. The Huber loss on log-residuals is minimized,

$$\mathcal{L}_{\text{fit}}(\mathcal{L}_\infty, A_P, A_D, \alpha, \beta) = \sum_i \text{Huber}_\delta\big(\log \mathcal{L}(P_i, D_i) - \log \mathcal{L}_i\big), \tag{28}$$

with $\delta = 10^{-3}$, using L-BFGS-B with grid-search initialization. The irreducible loss $\mathcal{L}_\infty$ and the data exponent $\alpha_D = \alpha/(\alpha+\beta)$ are reported from this fit. The IsoFLOP estimates of Step 2 remain the primary α_P and α_L values; the joint fit contributes the irreducible-loss term and the data exponent α_D.

Confidence intervals. Confidence intervals on the fitted parameters come from a residual bootstrap on the fit of Equation 29, matching the procedure of Section 8.2.1. 90% intervals are reported over 10,000 iterations, with intervals on the irreducible loss e tabulated in Table 7.

8.2.2 Results

Figure 22 shows the per-slice parabolic fits for each of the four families. Markers show measured validation losses, and $\times$ markers locate the per-slice minima $P^*(C)$. The fits are well-behaved across all four families: the parabolas tighten with increasing compute, and the per-slice minima move smoothly toward larger P as C grows.

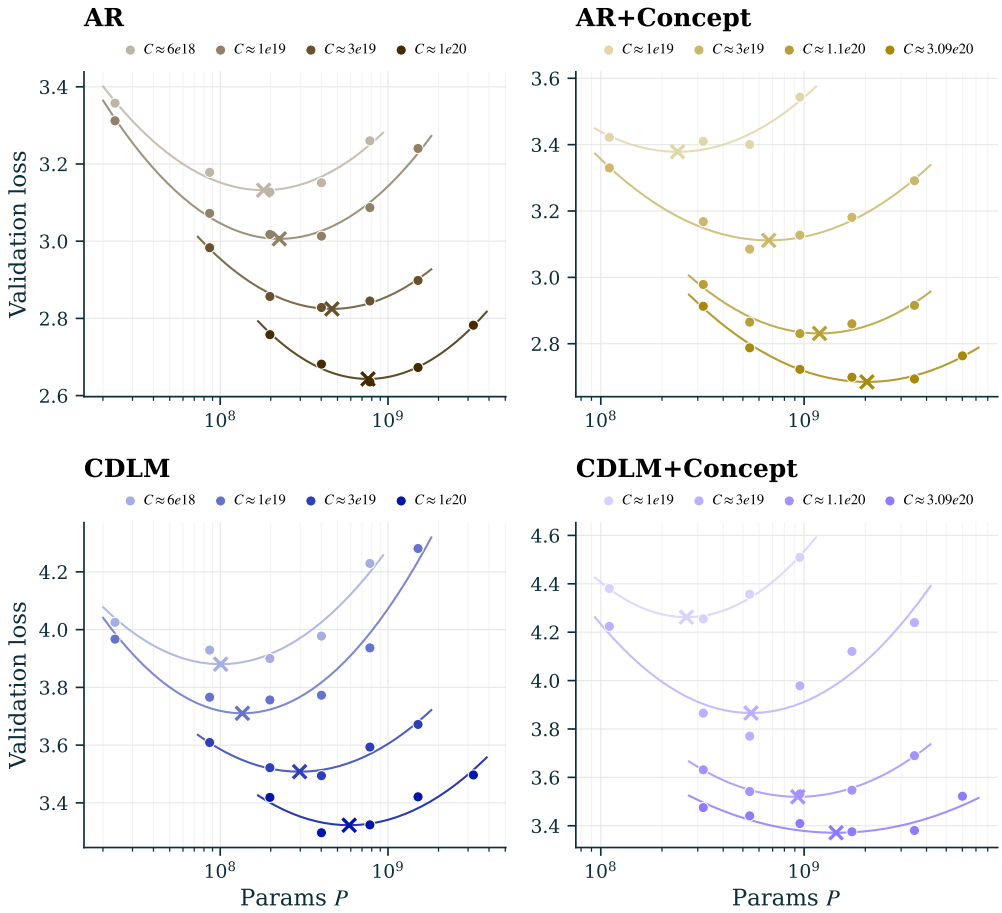

Figure 22. IsoFLOP analysis for each family. Each curve fits a parabola to model sizes within a fixed compute target C; markers show measured validation losses; $\times$ markers locate the per-slice minima $P^*(C)$.

The resulting power-law fits across compute are shown in Figure 23, with exponents and irreducible-loss asymptotes for all four families reported in Table 6. The autoregressive baseline gives $\alpha_P = 0.528$, in line with prior estimates of 0.49, 0.464, and 0.524 reported by Hoffmann et al. (2022), Shuai et al. (2024), and Bi et al. (2024) respectively. The fit also yields $\mathcal{L}_\infty = 1.857$, comparable to Chinchilla's 1.69. The diffusion baseline gives $\alpha_P = 0.632$, on the high end of prior masked-diffusion estimates of 0.514, 0.566, and 0.634 reported by Ni et al. (2025), von Rütte et al. (2025), and Nie et al. (2024) respectively. The higher value may result from the causal block attention mask (Section 5.2). Validation is reported in ELBO to follow prior work, although the choice of validation loss estimator can dramatically affect the fitted $\mathcal{L}^*$ values (Appendix H). von Rütte et al. (2025) additionally reports

α_P ranging from 0.535 to 0.589 across diffusion families and notes that the validation-loss ranking these exponents induce does not match the downstream performance ranking. The CDLM baseline yields $\mathcal{L}_\infty = 2.658$, modestly above Quokka's 2.41 but within the variation attributable to the ELBO estimator choice. The goal of this section is not to rank diffusion families against each other, but to measure the effect of adding the concept module on top of a fixed backbone trained on the same data; the comparisons that follow are therefore within-pair rather than cross-family.

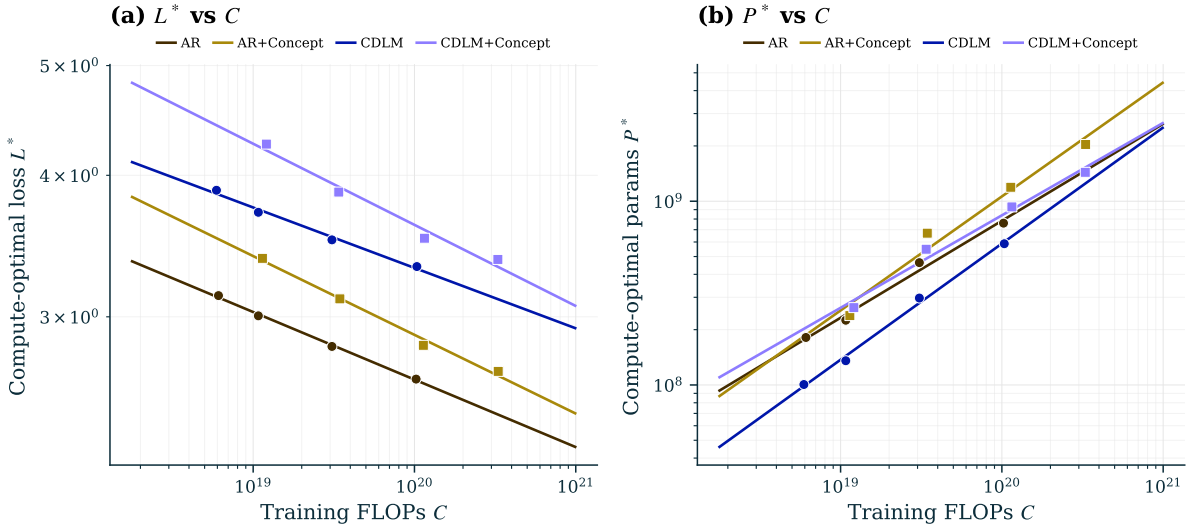

Figure 23. Compute-optimal scaling laws for each family. Panel (a): $\mathcal{L}^*(C)$ from the Step 2 power-law fits. Panel (b): $P^*(C)$ from the Step 2 power-law fits. Markers show per-slice minima from Figure 22.

Adding the concept module shifts the compute-optimal scaling exponents by a small, fixed per-backbone offset. For both pairs, α_L becomes more negative when the concept module is added. This direction is consistent with the fixed-library bias: the smallest +Concept models are over-parameterized by the concept library and under-utilize their parameters, biasing their losses upward and steepening the apparent loss-vs-compute slope. The shifts on α_P are small in both pairs. The irreducible-loss asymptote $\mathcal{L}_\infty$ shifts downward in both pairs ($1.857 \to 1.193$ for AR, $2.658 \to 1.942$ for CDLM); within-pair $\mathcal{L}_\infty$ comparisons are valid since both members of each pair use the same loss methodology.

Interpretability-by-design imposes a small per-backbone offset on compute-optimal scaling, not a scaling tax.

Predicting Steerling-8B from small models. The fitted exponents for the +Concept families are likely steeper than the asymptotic scaling rate, because the fixed concept library makes the smallest +Concept models artificially under-utilized and biases their losses upward. The joint Chinchilla form (Equation 27) accommodates this curvature by absorbing it into the irreducible loss $\mathcal{L}_\infty$; a naive log-linear extrapolation does not. Figure 24 refits both forms on the small-scale checkpoints only (excluding Steerling), and extrapolates to $C = 7.6 \times 10^{22}$ FLOPs, the compute used to train Steerling at 8B parameters and 1.35T tokens. The Steerling-8B model achieves $\mathcal{L}^* = 2.72$, while the joint Chinchilla fit predicts $\mathcal{L}^* = 2.61$, a gap of 0.11 nats. A naive log-linear extrapolation predicts

Model	α_P	α_D	α_L	$\mathcal{L}_\infty$
Ours (Autoregressive)				
AR	$0.528_{[-0.025,+0.023]}$	$0.445_{[-0.036,+0.114]}$	$-0.060_{[-0.002,+0.002]}$	$1.857_{[-0.335,+0.071]}$
AR+Concept	$0.621_{[-0.050,+0.086]}$	$0.524_{[-0.089,+0.093]}$	$-0.070_{[-0.002,+0.003]}$	$1.193_{[-0.580,+0.163]}$
Ours (Causal Diffusion)				
CDLM	$0.632_{[-0.091,+0.075]}$	$0.481_{[-0.111,+0.162]}$	$-0.053_{[-0.004,+0.004]}$	$2.658_{[-0.708,+0.181]}$
CDLM+Concept	$0.503_{[-0.061,+0.095]}$	$0.374_{[-0.187,+0.133]}$	$-0.072_{[-0.002,+0.002]}$	$1.942_{[-0.940,+0.159]}$
Masked diffusion (prior)				
Ni et al. (2025)	0.514	0.486	—	2.41
von Rütte et al. (2025)	$0.566_{[-0.022,+0.019]}$	$0.434_{[-0.019,+0.020]}$	$-0.0496_{[-0.0004,+0.0003]}$	—
Autoregressive (prior)				
Hoffmann et al. (2022)	0.490	0.510	—	1.69
Shuai et al. (2024)	0.464	0.536	—	—
Bi et al. (2024)	0.524	0.476	—	—

Table 6. Compute-optimal scaling exponents and irreducible-loss asymptotes. Subscripts are 90% bootstrap confidence intervals.

$\mathcal{L}^* \approx 2.25$, missing by 0.47 nats. The joint fit's prediction error is roughly $4\times$ smaller than the log-linear extrapolation's.

Inherently interpretable architectures admit scaling-law extrapolation. Steerling-8B's validation loss is predicted from small-scale fits to within 0.11 nats via the joint Chinchilla form.

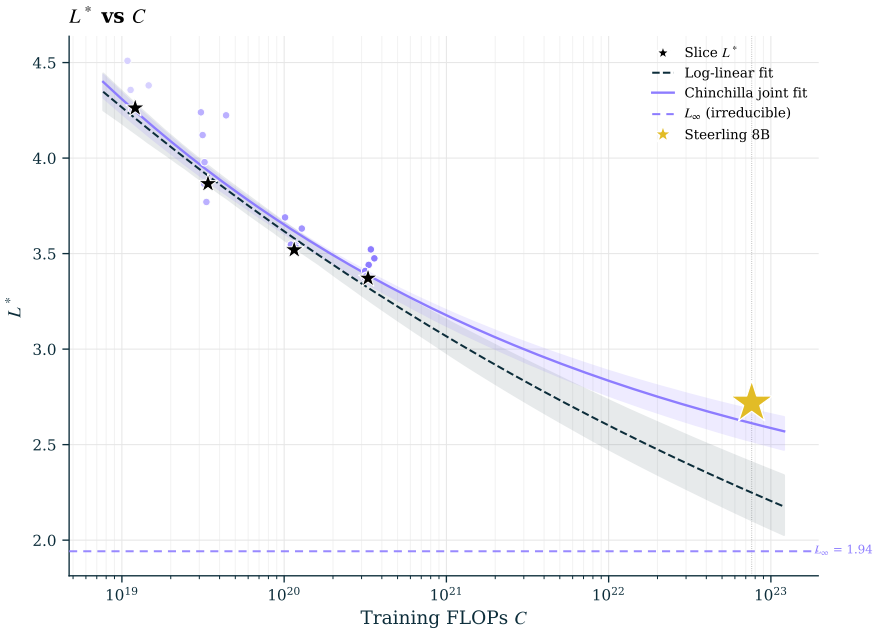

Figure 24. Extrapolating CDLM+Concept to Steerling. Joint Chinchilla fit (solid, $\mathcal{L}_\infty$ dashed) and log-linear fit (dashed) on slice $\mathcal{L}^*$ minima. Steerling (filled star) lands within 0.11 nats of the joint fit.

8.3 Interpretability scaling

Section 8.2 established that adding the concept module preserves compute-optimal scaling: interpretability-by-design imposes at most a fixed per-backbone offset on validation loss. The question that remains is how interpretability itself behaves at scale. The metrics defined in Section 7 are fit against training compute on the same checkpoints used in Section 8.2, allowing a direct test of **RQ2: does interpretability scale favorably with compute?**

8.3.1 Methodology

Metrics. Four metrics from Section 7 are fit against training compute: Concept Loss (↓), Concept Independence Loss (↓), Concept Contribution (↑), and Known Concept Alignment (↑).

Scaling-law fit. Unlike validation loss, interpretability metrics are not analyzed under a compute-optimal frontier; they improve monotonically with compute. Each metric is therefore fit directly against training compute on all checkpoints rather than on per-slice minima. The metrics are bounded by construction (Concept Loss and Concept Independence Loss below by zero, Concept Contribution above by one, Known Concept Alignment above by five), and a log-linear fit ignores these bounds, predicting values outside them when extrapolated past the data range. The form used here, adapted from the irreducible-loss scaling laws of Hoffmann et al. (2022) and applied to SAE interpretability by Gao et al. (2025), encodes the bound directly as a free parameter:

$$m(C) = e \pm A\,C^{-\beta}, \tag{29}$$

with three free parameters A, β, e fit jointly by nonlinear least squares, where e is the irreducible loss the metric approaches at infinite compute. The sign is positive for ↓ metrics (approaching e from above) and negative for ↑ metrics (approaching e from below).

Confidence intervals. Confidence intervals on the fitted parameters come from a residual bootstrap on the fit of Equation 29, matching the procedure of Section 8.2.1. 90% intervals are reported over $10{,}000$ iterations, with intervals on the irreducible loss e tabulated in Table 7.

8.3.2 Results

Figure 25 shows the fitted curves for the four metrics on both +Concept families, with fitted parameters and bootstrap confidence intervals on the irreducible loss e reported in Table 7. Every metric improves in the expected direction on both backbones: Concept Loss and Concept Independence Loss decrease with compute, while Concept Contribution and Known Concept Alignment increase. The fits explain a modest fraction of the per-checkpoint variance (R^2 between 0.50 and 0.73), reflecting the noise in evaluating interpretability properties on individual checkpoints. For several metrics the fitted asymptote is pinned to its natural bound, with bootstrap intervals concentrated near the boundary; this indicates that the small-scale data does not yet curve enough to identify e from the data range alone, and the asymptote estimates should be read as upper or lower bounds on the true plateau rather than precise predictions.

Across both backbones, all four metrics improve with compute under the asymptotic form. Concept Loss falls toward zero, indicating that the concept module continues to identify concepts more accurately as training scales. Concept Independence Loss falls similarly, indicating that the known and unknown pathways become more disentangled. Concept Contribution rises toward one, indicating that predictions route through the concept module rather than the residual at increasing fractions.

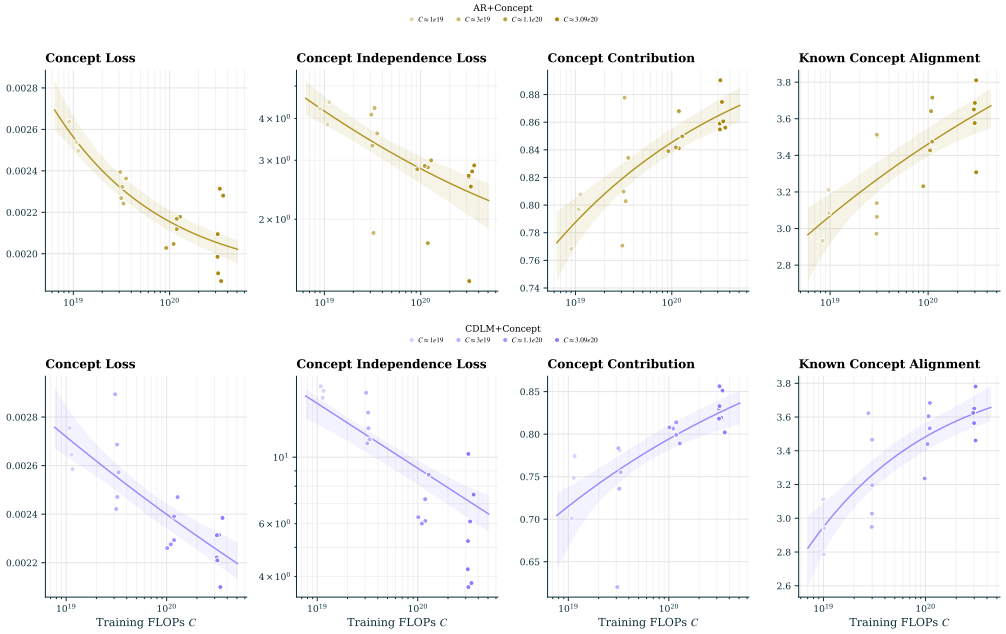

Figure 25. Interpretability metrics scaling with compute, fit using the power-law-with-irreducible-loss form (Equation 29). Top row: AR+Concept. Bottom row: CDLM+Concept. Markers colored by IsoFLOP slice; shaded bands are 90% bootstrap confidence intervals on the fitted curve.

Family	β	e [90% CI]	R^2	8B pred	8B actual	Δ
Concept Loss						
AR+Concept	0.385	$0.002_{[-0.002,+0.000]}$	0.704	0.002	—	—
CDLM+Concept	0.054	$0.000_{[-0.000,+0.002]}$	0.648	0.002	0.002	+0.000
Concept Independence Loss						
AR+Concept	0.279	$1.310_{[-1.310,+0.000]}$	0.505	1.550	—	—
CDLM+Concept	0.216	$0.000_{[-0.000,+3.690]}$	0.725	2.190	1.550	−0.640
Concept Contribution						
AR+Concept	0.212	$0.937_{[-0.047,+0.063]}$	0.630	0.915	—	—
CDLM+Concept	0.141	$1.000_{[-0.144,+0.000]}$	0.522	0.919	0.876	−0.044
Known Concept Alignment						
AR+Concept	0.099	$5.000_{[-1.190,+0.000]}$	0.557	4.200	—	—
CDLM+Concept	0.349	$3.920_{[-0.135,+1.080]}$	0.637	3.870	3.770	−0.100

Table 7. Interpretability scaling fits. Subscripts on e are 90% bootstrap confidence intervals. The 8B columns compare the small-scale extrapolation against the actual Steerling values.

Known Concept Alignment rises toward saturating at the judge ceiling, indicating that concept embeddings increasingly point at semantically related tokens.

Interpretability scales favorably with compute across both autoregressive and diffusion.

Predicting Steerling-8B from small models. The fits are now refit on the small-scale checkpoints only and extrapolated to $C = 7.6 \times 10^{22}$ FLOPs, the compute used to train Steerling at 8B parameters and 1.35T tokens. Figure 26 compares the extrapolations to the deployed model's measured metrics, and the rightmost columns of Table 7 report the predicted and actual values per metric. Three of the four metrics land within tight bounds of the small-scale extrapolation: Concept Loss within 0.0004 on a BCE scale, Concept Contribution within 0.05 on a $[0, 1]$ scale, and Known Concept Alignment within 0.10 on a 1-5 scale. Concept Independence Loss exceeds the small-scale extrapolation in the favorable direction, with the deployed model achieving 1.55 against a predicted 2.19, consistent with the architecture continuing to disentangle its known and unknown pathways at scale beyond what the small-scale fit captures.

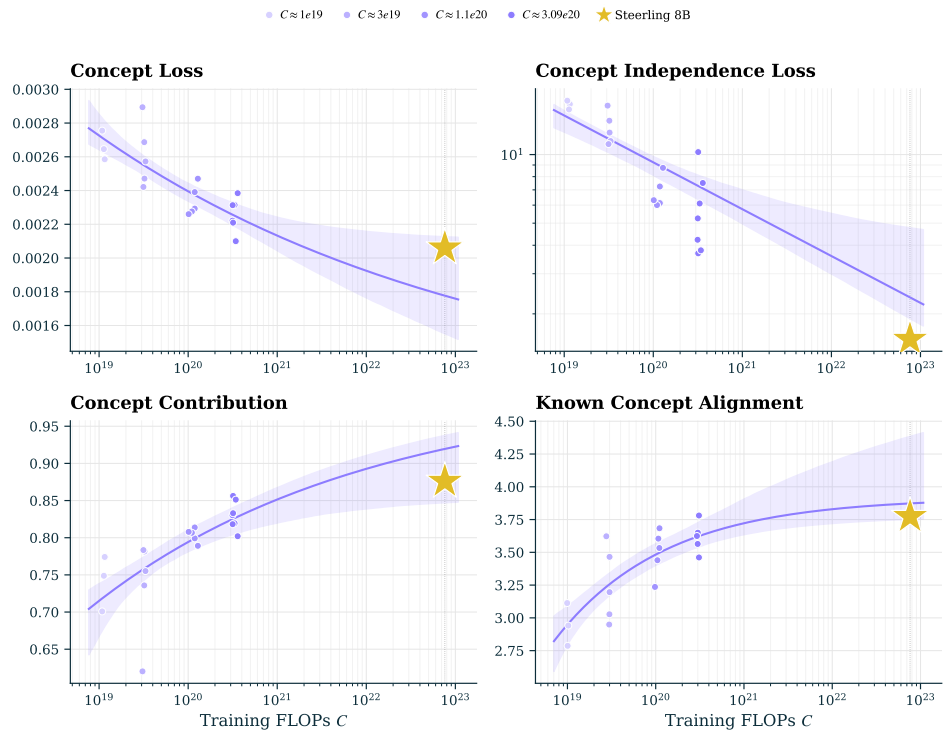

Figure 26. Extrapolating CDLM+Concept interpretability metrics to Steerling. Asymptotic fits on small-scale checkpoints (open markers, colored by IsoFLOP slice) with 90% bootstrap CI bands. Filled stars: Steerling at 8B parameters and 1.35T tokens.

Inherently interpretable architectures admit scaling-law extrapolation for interpretability properties. The deployed Steerling-8B model is predicted within error bounds across all four metrics from small-scale fits.

9 Steerling-8B: Pretraining

With the recipe established in Section 3 and its scaling behavior characterized in Section 8, this section describes the full-scale pretraining of Steerling-8B. We cover the concept-annotated 1.2T-token dataset (Section 9.1), the architectural and training choices that distinguish Steerling-8B from a standard autoregressive or diffusion backbone (Section 9.2), the pretraining run itself (Section 9.3), and the lessons we learned (Section 9.4).

9.1 Data

We pretrain Steerling-8B on 1.2T tokens drawn from a mixture of high-quality web, academic, mathematical, and code corpora. The bulk of the mix is Nemotron-CC-HQ (Su et al., 2025), a quality-filtered slice of Nemotron-CC composed of both real webtext and synthetic question-and-answer rephrasings generated from the same documents. We choose Nemotron-CC-HQ because it yields the strongest downstream performance among public 1T pretraining datasets. The remainder of the corpus consists of peS2o (Soldaini and Lo, 2023), arXiv (Weber et al., 2024), OpenWebMath (Paster et al., 2024a), Algebraic Stack (Azerbayev et al., 2023b), StarCoder (Li et al., 2023), and Wikipedia and Wikibooks following the OLMo 2 mixture (Team OLMo et al., 2024). We annotate the corpus with concepts at the chunk level using the Atlas pipeline (Section 4). The full per-source token counts are reported in Table 8.

Source	Documents	Chunks	Tokens
Nemotron-CC-HQ (real)	740M	5.1B	547B
Nemotron-CC-HQ (synthetic)	971M	4.8B	498B
peS2o	38.8M	565.4M	59B
arXiv	3.9M	142.2M	20.4B
Wikipedia & Wikibooks	6.1M	36.8M	3.8B
OpenWebMath	2.9M	76.8M	12.1B
Algebraic Stack	2.8M	65.6M	12.1B
StarCoder	78.6M	317M	91.4B
Total	**1.84B**	**11.1B**	**1.24T**

Table 8. Steerling-8B pretraining corpus. Token counts are post-tokenization.

9.2 Pretraining recipe

Training a standard language model requires choosing a handful of hyperparameters such as sequence length, batch size, learning rate, optimizer, and weight decay. The community has well-established defaults for each, and we adopt them without modification for Steerling-8B. However, our interpretable causal-diffusion architecture (Section 5) introduces design decisions without precedent in the autoregressive or masked-diffusion literature, for which no community defaults yet exist.

These fall into two groups: those that govern the diffusion process (Section 5.2) and those that govern the concept bottleneck (Section 5.3). We describe each choice and the reasoning below; for every choice we ran a small ablation at the 1B scale, reported in full in Appendix J.

Diffusion process. Steerling-8B uses the block-causal attention mask of Section 5.2, where tokens attend bidirectionally within a block of size b and causally across blocks. The block size sets how

many tokens are decoded in parallel at inference and the granularity of the key-value cache. We compare $b \in \{32, 64\}$ and find block size leaves the interpretability metrics unchanged while the larger value lowers validation loss, so we set $b = 64$. Diffusion training must also choose how the noise level t is sampled at each step. We compare uniform sampling against a moving Gaussian curriculum that shifts from low to high masking over training (Ni et al., 2025), and adopt the moving Gaussian for a slight edge on validation loss, consistent with prior work.[2]

Concept bottleneck sizing. The concept module splits its representation into n known concepts, fixed by the Atlas library, and m unknown concepts learned during training. Here m is a free hyperparameter. We find that raising it from $3n$ to $5n$ gives no measurable gain on any metric, so we keep the conservative $m = 3n$. The unknown embedding matrix $U \in \mathbb{R}^{m \times d}$ is the largest parameter the module adds; we factorize it as $U = AB$ with rank $R = 256$, which makes it roughly $15\times$ smaller and removes a heavy per-token matrix multiplication at no capability cost and only a small drop in concept contribution.

Concept bottleneck training dynamics. Both concept heads are trained from scratch, so their early predictions are unreliable. We therefore route ground-truth concepts to the LM head early and anneal toward the model's own predictions, holding the teacher forcing floor at 0.5 for both heads, since decaying further hurts known concept alignment for no capability gain. The model can optionally carry a residual $\varepsilon = h - \hat{k} - \hat{u}$ in the bottleneck, an uninterpreted channel that absorbs whatever the two heads fail to reconstruct. Dropping it forces every dimension through the known and unknown heads, raising concept contribution to 1.0 by construction, but in our comparison the capability gap is large enough that we keep ε.

9.3 Pretraining run

Pretraining used the configuration of Appendix K on 320 A100 GPUs (40 nodes, 8 GPUs per node) for approximately 21 days, totaling ∼161K GPU-hours. The run completed its full 1.2T-token budget without divergence or manual intervention to the loss curve.

We monitored various metrics during the pretraining run. Validation loss was logged continuously, computed as a Monte Carlo estimate of the MDLM ELBO (Section 8.2.1). At every 50B-token interval, we additionally saved a checkpoint and ran a broader evaluation suite of five downstream benchmarks from the language modelling harness: ARC-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), MMLU (Hendrycks et al., 2020), and WinoGrande (Sakaguchi et al., 2021); together with the four interpretability metrics introduced in Section 7: concept loss, concept contribution, concept independence loss, and known concept alignment.

Figure 27 shows the capability metrics across the run. Validation loss descends rapidly through the first half and then plateaus, oscillating around a stable value. The harness benchmarks behave differently from one another. HellaSwag and PIQA rise smoothly and hold near their peaks to the end, and ARC-Challenge peaks late with only a small dip at the very end. MMLU and WinoGrande, by contrast, peak around the midpoint and then decline through the final third, with MMLU losing roughly a quarter of its peak value.

Figure 28 shows the interpretability metrics across the run. Concept loss drops early and then climbs steadily, though the climb is small in absolute terms. Concept independence loss stays low through most of training, then rises sharply in the final third, peaking around 2.7. Concept contribution dips

[2]This schedule proved too aggressive over the full pretraining run (see Section 9.4.1).

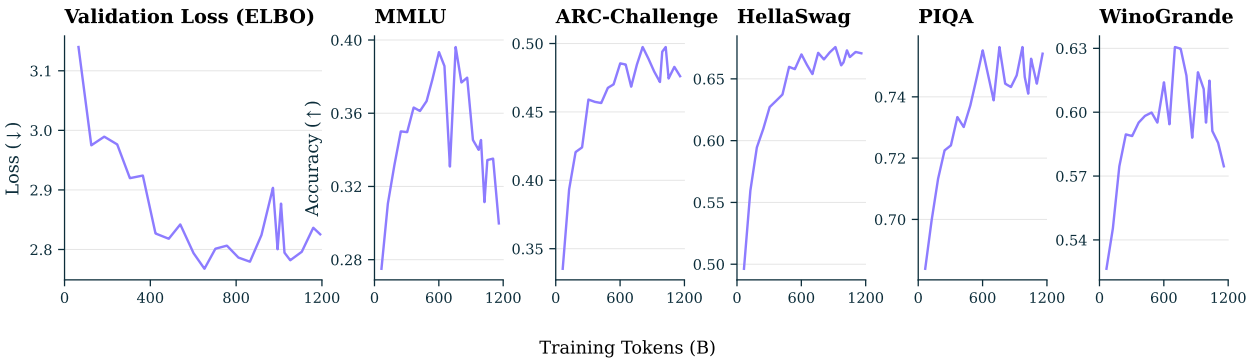

Figure 27. Validation loss and LM-Harness metrics for Steerling-8B across the pretraining run.

slightly early before climbing from 0.62 to 0.85. Known concept alignment moves modestly on its $[1, 5]$ scale, rising to a peak near 4 in the first third before settling back.

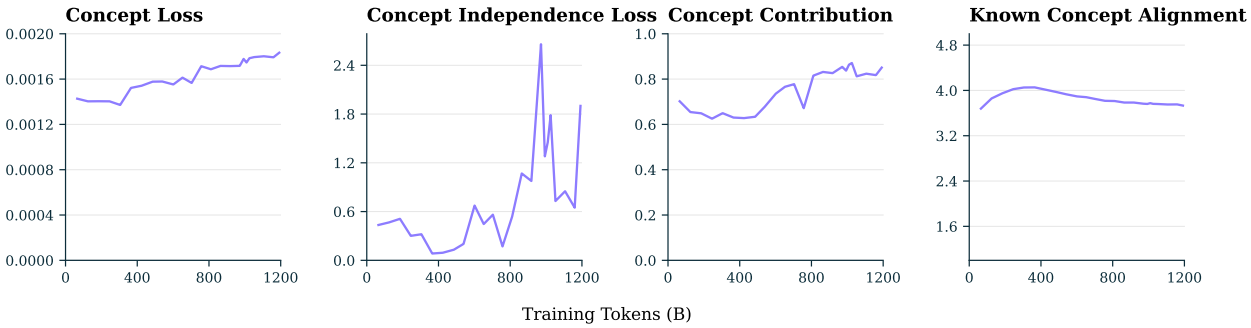

Figure 28. Interpretability metrics for Steerling-8B across the pretraining run.

9.4 Pretraining lessons

During the run, most metrics looked healthy. Validation loss kept descending or held steady, three of the four interpretability metrics drifted only modestly, and HellaSwag, PIQA, and ARC-Challenge plateaued near their peaks. However, four signals were less encouraging:

- MMLU and WinoGrande began declining around the midpoint of training.
- Concept independence loss spiked in the final third.
- A late checkpoint showed weak math and code performance.
- The same checkpoint did not steer reliably through the concept module.

At the time we expected some of these to recover as the schedules stabilized, so we continued the run to its planned 1.2T token budget. Once the run completed, we examined the issues closely and traced them to four causes: an over-aggressive masking curriculum (Section 9.4.1), late-training entanglement of the concept heads (Section 9.4.2), insufficient math and code in the corpus (Section 9.4.3), and the absence of steering operations during training (Section 9.4.4).

9.4.1 Over-aggressive masking

MMLU and WinoGrande peak around the midpoint of training and decline thereafter. Overlaying the masking curriculum on the capability metrics (Figure 49) makes the cause visible: both inflection points coincide with the curriculum reaching ~ 0.5 on its climb to its endpoint of 0.8. The moving Gaussian curriculum was validated at small scale (Appendix J.2), where it held a slight edge on validation loss. At pretraining scale, however, the curriculum spends a substantial fraction of training above 50% masking, where most of the input is replaced with `[MASK]` tokens and the model has little context to leverage.

9.4.2 Late-training entanglement of the concept heads

Concept independence loss stays low through most of the run and rises sharply in the final third. Overlaying the three schedules on the interpretability metrics shows why (Figure 50). Two of them move into adverse territory over the same window. The masking curriculum reaches its hard regime, so most input tokens are masked. At the same time the teacher forcing floors have been reached, with α_{known} at its floor and α_{unknown} decayed substantially, so the LM head leans increasingly on predicted concepts rather than ground-truth ones. With less supervision and more reliance on predicted contributions, the two heads come to rest on overlapping information, and disentangling their representations becomes harder. At ablation scale this effect was not visible.

9.4.3 Limited math and code in the corpus

Our corpus is dominated by Nemotron-CC-HQ, which carries little math or code. Together with the smaller dedicated sources in the mix (Table 8), the run saw slightly over 100B math and code tokens out of 1.2T total. von Rütte et al. (2025) report a similar shortfall on the same corpus. We caught this only at the end of pretraining, when GSM8K and HumanEval both came back low. With more math and code in the mixture, and these benchmarks included in the in-run evaluation sweep, the issue would have surfaced earlier and been addressable on the data side.

9.4.4 The model does not respond to steering

A model built around an explicit concept module should be steerable through it, since injecting a concept's own direction (Section 6.2) should bias generation toward that concept. We tested this on a late pretraining checkpoint following the evaluation protocol of Wu et al. (2025). We elicit 128-token generations from a fixed prompt, sampling a continuation under the method being evaluated, and score each continuation with an LLM-judge on two axes. A *concept* score (0–2) measures how strongly the target concept is present in the generated text, and a *quality* score (0–2) measures the fluency and coherence of the output. Details on the judge and the full prompts are given in Appendix M. Finally, we summarize the two scores by their harmonic mean:

$$\text{Harmonic} = \frac{2}{\frac{1}{\text{concept}} + \frac{1}{\text{quality}}}, \tag{30}$$

which weights both axes equally and strongly penalizes a method that sacrifices one for the other. We report the mean over 72 randomly sampled concepts.

We study steering on three conditions: *unsteered*, *prompting* (via prepending the concept label and description to the prompt), and *steered* generation (layer injection in Section 6.2). In Table 9, unsteered generation produces fluent text that does not surface the target concept, which is expected. Steering achieves the highest concept score of 1.072, showing that intervening through the model's

own concept direction induces the target concept more strongly than though a text prompt. Yet, the cost of steering is quality. As a result, prompting yields the best harmonic mean of 1.156.

Method	Concept ↑	Quality ↑	Harmonic ↑
Unsteered	0.033	1.108	0.065
Prompting	0.908	**1.588**	**1.156**
Steered	**1.072**	0.972	1.020

Table 9. Steering results on random concepts of the pretrained Steerling-8B. *Steered* denotes layer injection.

From this experiment, we identify two limitations for steering. First, the quality drop under steering is substantial, suggesting Steerling-8B does not gracefully integrate the injected direction into its forward pass. Second, layer injection does not generalize on less-frequent concepts, as roughly one third never activate at any injection strength γ, with the bottleneck activation k_c staying near zero throughout generation. Both trace to the same cause, that Steerling-8B never encounters concept injection during pretraining, so adding $\gamma \cdot K_c$ at inference is an out-of-distribution perturbation the model has no mechanism to respond to.

10 Steerling-8B: Mid-training

Mid-training is a short 150B-token run initialized from the final pretraining checkpoint. It has two aims. First, address the four weaknesses identified in pretraining (Section 9.4): aggressive masking, thin math and code coverage, unstable independence loss, and no exposure to steering during training. Second, tighten the model for downstream use, by reducing its reliance on teacher forcing and increasing its reliance on the concept heads. Here we describe the data mixture (Section 10.1), the recipe changes from pretraining including a dedicated steering phase (Section 10.2, Section 10.2.4), the resulting mid-trained model (Section 10.3), and its benchmark performance against open base models of comparable size (Section 10.4).

10.1 Data

Our goal in mid-training is to improve Steerling-8B on math and code, where the pretrained model lagged furthest behind, and to recover the reasoning and knowledge capabilities degraded by heavy masking late in pretraining. We follow the OLMo 2 mid-training recipe (Team OLMo et al., 2024): start from a high-quality data mixture that lifts performance across the benchmark suite, then patch the specific capabilities the pretrained model is weakest on. For the natural-language portion of the mixture we again use Nemotron-CC-HQ, but restrict to real tokens; we benchmarked real, synthetic, and mixed against each other (Appendix N.1), and real tokens won.

To find the best midtraining mixture, we run a data ablation: starting from the final pretraining checkpoint of 1.2T tokens, we midtrain on 10B tokens for each of four candidate compositions and compare their downstream performance. The compositions, listed in Table 10, are a math-heavy mixture, a balanced mixture following OLMo 2's Dolmino Mix, a code-augmented mixture that adds StarCoder to the balanced mixture, and a code-only mixture. The first three hold the Nemotron-real share roughly fixed and vary the math, code, and high-quality reference sources around it; the code-only mixture is an extreme that drops Nemotron entirely, isolating the effect of training on code alone.

Source	Math-heavy	Balanced	Code-augmented	Code-only
Nemotron (real)	5.0B	4.7B	5.0B	–
Dolmino Math	5.0B	2.1B	2.0B	–
StarCoder	–	–	1.0B	10.0B
FLAN	–	1.7B	1.0B	–
peS2o	–	0.6B	0.4B	–
Wikipedia & Wikibooks	–	0.7B	0.5B	–
Stack Exchange	–	0.2B	0.2B	–
Total	10.0B	10.0B	10.0B	10.0B

Table 10. Data compositions compared in the mid-training ablation. Each arm is a 10B-token run from the final pretraining checkpoint of 1.2T tokens; entries are token counts in billions.

Composition	MMLU	GSM8K	ARC-C	HSwag	HEval	MBPP	WinoG	Avg.
Pretrained model	0.298	0.140	0.484	0.673	0.049	0.004	0.596	0.321
Math-heavy	0.376	**0.441**	0.492	0.681	0.037	**0.012**	0.616	**0.379**
Balanced	**0.416**	0.328	0.497	**0.693**	0.037	0.006	0.616	0.371
Code-augmented	**0.416**	0.328	**0.498**	**0.693**	0.055	**0.012**	**0.628**	0.376
Code-only	0.303	0.086	0.434	0.630	**0.061**	**0.012**	0.583	0.301

Table 11. Downstream performance of the four mid-training compositions, each a 10B-token run from the final pretraining checkpoint, against the base model. Best in each column in bold. HSwag: HellaSwag; HEval: HumanEval; WinoG: WinoGrande.

The results are shown in Table 11. Every mixture except code-only **improves substantially** over the pretrained model, but each isolates a different lesson:

- The math-heavy mixture delivers the largest math gain, lifting GSM8K from 0.140 to 0.441, and posts the best overall average, but this is driven by large gains on only two benchmarks while it lags on the rest.
- The balanced mixture drives the largest knowledge gain, recovering MMLU from 0.298 to 0.416, but with no dedicated code source, HumanEval again fails to improve.
- The code-only mixture pushes HumanEval the highest, as expected, but lags well behind every other composition elsewhere.
- The code-augmented mixture strikes a balance across reasoning, math, and code, and is the only composition to improve on every benchmark over the pretrained model, so we adopt it as the mid-training mixture.

The final midtraining mixture applies the code-augmented recipe over 150B tokens, with proportions given in Table 12.

10.2 Mid-training recipe changes

Source	Number of Tokens	Ratio (%)
Nemotron (real)	72.79B	48.5
StarCoder	30.75B	20.5
peS2o	21.98B	14.7
Dolmino Math ($\sim$2$\times$)	16.05B	10.7
FLAN	6.38B	4.3
Wikipedia & Wikibooks	1.50B	1.0
Stack Exchange	0.56B	0.4
Total	150B	100.0

Table 12. The final Steerling-8B midtraining mixture of 150B tokens. *Dolmino Math* is upsampled roughly twofold. Proportions follow the code-augmented composition of Table 10.

10.2.1 Schedules

Masking schedule. The first and most consequential change is to revert the masking schedule. Pretraining used a moving Gaussian curriculum whose center rose from 0.2 to 0.8, and Section 9.4 traced Steerling-8B's declining knowledge and reasoning scores to the high-masking regime this curriculum entered late in the run. Midtraining instead samples the masking rate uniformly, following MDLM training (Nie et al., 2025), so the model sees a balanced spread of masking levels rather than a curriculum that drifts toward heavy masking. To confirm that uniform sampling is the right choice at this stage, we compare it against an 80% Gaussian schedule in a 10B-token ablation from the final pretraining checkpoint (Table 13). The two schedules are close on most benchmarks, and the 80% schedule is even slightly stronger on the math tasks. On MMLU, however, the heavy schedule falls to 0.280, below the pretrained model's 0.298, while uniform sampling lifts it to 0.416; the heavy-masking regime that hurt knowledge during pretraining hurts it again here, confirming our initial suspicion. We adopt a uniform schedule during midtraining.

Masking	MMLU	GSM8K	ARC-C	HSwag	HEval	MBPP	WinoG	Avg.
Pretrained model	0.298	0.140	0.484	0.673	0.049	0.004	0.596	0.321
50% uniform	**0.416**	0.328	0.498	**0.693**	**0.055**	**0.012**	**0.628**	**0.376**
80% Gaussian	0.280	**0.355**	**0.500**	0.686	**0.055**	0.008	**0.628**	0.359

Table 13. Masking schedule ablation, each a 10B-token run from the final pretraining checkpoint. Best in each column in bold. HSwag: HellaSwag; HEval: HumanEval; WinoG: WinoGrande.

Teacher forcing. During pretraining the model's predicted concepts were partly replaced with ground-truth ones: with probability α_{known} the known head's predicted activations were replaced by their labeled values, and the unknown head was also mixed with its residual at rate α_{unknown}. Mid-training anneals α_{known} from its end-of-pretraining value of 0.5 to 0, and holds α_{unknown} at 0 throughout. By the end of mid-training, every concept entering the bottleneck is model-predicted, matching the inference regime where labeled concepts are unavailable.

Learning rate. Following standard mid-training practice (Team OLMo et al., 2024; Grattafiori et al., 2024a; Liu et al., 2024), the learning rate is decayed linearly to zero.

10.2.2 Losses

The second pretraining weakness was the independence loss (Equation 14), which destabilized late in the run as the model came to rely on its predicted concepts (Section 9.4). Midtraining makes the model rely on the predicted concepts entirely, which exposes a limitation of the single-term loss. The pretraining loss penalized the dependence between the known representation $\hat{k}$ and the unknown representation routed to the bottleneck, but with the unknown head now always supplying its own prediction, detached before the bottleneck (gradient detachment), the term applies no pressure on the transformer residual. We therefore replace it with two terms, $\mathcal{L}_{\text{indep}}(\hat{u}, \hat{k}) + \mathcal{L}_{\text{indep}}(\varepsilon, \hat{k})$: the first keeps the predicted unknown independent of the known concepts, the second keeps the residual ε independent of them. Each term carries half the original weight, leaving the total magnitude unchanged.

10.2.3 Architecture

Two architectural changes tighten interpretability. First, the residual dropout rate p_ε is raised from 0.1 to 0.3, applying more pressure on ε to vanish and forcing the concept heads to carry a larger share of the prediction. Second, the concept heads are sparsified: the known head composes the bottleneck from its top 32 concepts, and the unknown head from its top 128. Sparsity makes attribution more interpretable, since each logit decomposes into at most 160 concepts rather than the full bottleneck of thousands.

10.2.4 Steering training

Pretraining labels mark that a concept appears in a chunk but not which tokens actually express it. Steering supervision needs that detail because injection happens at the token level: at each generation step, we add a concept embedding into the transformer's hidden state at each token position, so the loss must know which positions should actually respond. We therefore mid-train on a token-level dataset (Figure 29) of roughly 400M tokens, where each token is tagged with its attributed concepts.

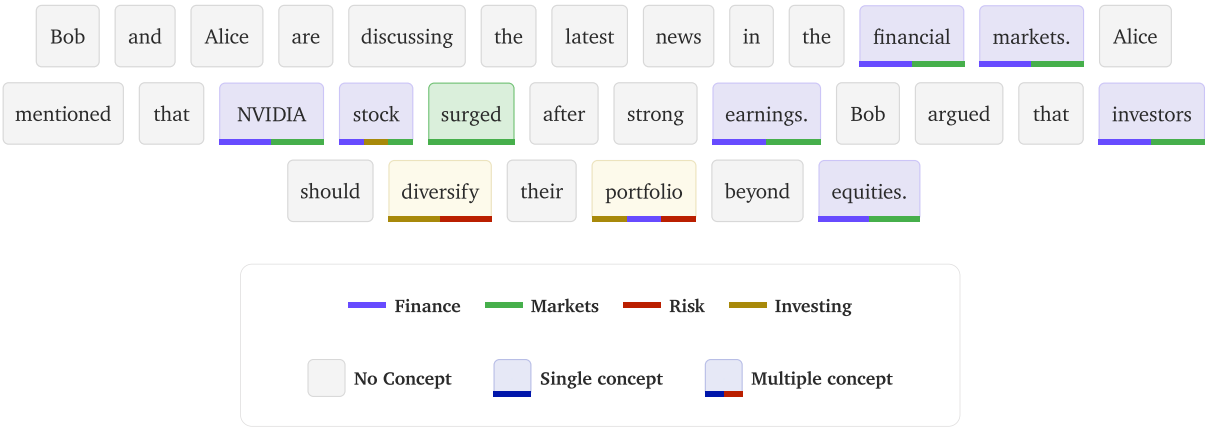

Figure 29. Token-level concept annotations used for steering training. Each token is tagged only with the concepts attributed to it, rather than with all chunk-level concepts in pretraining.

Steering losses. During a steering phase the target concept's direction K_c, scaled by γ, is injected at the masked positions attributed to c, exactly as steering is applied at inference. Let $\mathcal{I}$ denote the set of these injected positions. Two losses are added on top of the masked diffusion loss $\mathcal{L}_{\text{MDM}}$, one for each objective:

- **Respond objective.** At every injected position, the concept bottleneck should activate the injected concept. Its activation $k_{c,t}$ is driven toward 1 by minimizing its negative log-likelihood:

$$\mathcal{L}_{\text{respond}} = -\frac{1}{|\mathcal{I}|} \sum_{t \in \mathcal{I}} \log k_{c,t}. \tag{31}$$

- **Express objective.** A high bottleneck activation does not by itself produce concept-expressing output: the concept direction may still place weight on unrelated tokens such as fillers. We therefore push the concept's output distribution at attributed positions onto the tokens that express c.

  These are the lifted set $\mathcal{T}_c$: vocabulary tokens with the highest *lift*, defined as the ratio of token frequency in concept-tagged chunks to token frequency in the corpus. We maximize the total probability assigned to $\mathcal{T}_c$, i.e., minimize its negative log:

$$\mathcal{L}_{\text{express}} = -\frac{1}{|\mathcal{I}|} \sum_{t \in \mathcal{I}} \log \frac{\sum_{y \in \mathcal{T}_c} e^{\ell_{t,y}}}{\sum_{y \in V} e^{\ell_{t,y}}}, \tag{32}$$

  where $\ell_{t,y}$ is the logit of token y at position t.

The masked diffusion loss $\mathcal{L}_{\text{MDM}}$ is kept active throughout so that generation stays coherent, while the pretraining interpretability losses (the concept, reconstruction, and independence terms) are disabled during steering phases. The steering-phase objective is

$$\mathcal{L} = \mathcal{L}_{\text{MDM}} + \lambda_{\text{respond}} \mathcal{L}_{\text{respond}} + \lambda_{\text{express}} \mathcal{L}_{\text{express}}, \tag{33}$$

where λ_{respond} and λ_{express} are both set to 1.

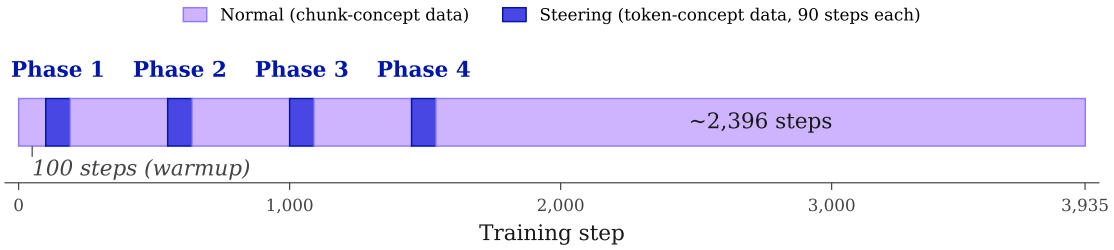

Figure 30. Interleaved mid-training schedule for Steerling-8B. Four steering phases (90 steps each, shaded) are inserted within the standard run; the first normal block is a 100-step warmup. Normal phases use chunk-level concept data, steering phases use token-level concept data.

Steering training schedule. Steering is interleaved into the main midtraining run rather than added as a separate stage (Figure 30). We front-load it: four short steering phases come immediately after a 100-step warmup, and the remainder of mid-training is normal capability training. This placement keeps the steering objectives from interfering with the capabilities the rest of the run builds. Each phase is one pass over the token-level dataset (∼400M tokens); we use four phases because both $\mathcal{L}_{\text{respond}}$ and $\mathcal{L}_{\text{express}}$ converge by the fourth epoch. Normal phases use chunk-level data and the standard objective; steering phases use token-level data and the steering losses above.

Steering training works and does not cost capability. We evaluate steering as an ablation on the mid-training recipe: starting from the math-heavy composition tested in Section 10.1, the 10B-token run is repeated with the steering phases interleaved, so that the only difference from the recipe in Table 11 is steering itself. Adding steering leaves LM Harness performance essentially unchanged across all five benchmarks (Table 14), so teaching the model to respond to injection does not trade off against capability. It also achieves its intended effect: the steering benchmark scores all improve, and the model activates the target concept in a single injection step at $\gamma = 1$.

	LM Harness ↑					Steering benchmark ↑		
	MMLU	GSM8K	ARC-C	HSwag	WinoG	Concept	Quality	Harmonic
Math-heavy	0.370	**0.431**	0.499	**0.682**	**0.618**	1.208	0.989	1.088
+ steering	**0.384**	0.415	**0.505**	0.681	0.611	**1.244**	**1.139**	**1.189**

Table 14. Steering ablation: Adding the steering phases improves every metric on the steering benchmark (Wu et al., 2025), while LM Harness performance stays mostly unchanged.

10.3 The mid-trained model

The final mid-training run is 150B tokens on the code-augmented mixture (Section 10.1), starting from the final pretraining checkpoint, with the recipe changes of Section 10.2: uniform masking, teacher forcing annealed to zero, the two-term independence loss, residual dropout raised to 0.3, sparsified concept heads, and a learning rate decayed to zero. Steering is interleaved into this run rather than added as a separate stage, as four short phases over a token-level dataset of roughly 400M tokens (Section 10.2.4). The complete hyperparameter list is given in Appendix N.2.

Mid-training rescues the capabilities lost in pretraining. Mid-training addresses the capability weaknesses of pretraining at once (Table 15). The knowledge lost under heavy masking comes back, with MMLU up **17 percentage points**. The two capabilities the pretraining corpus starved improve the most: math rises **30 percentage points** and code around **7** percentage points on average (HEval & MBPP). The benchmarks that were already healthy stay healthy, so nothing is traded away. **Every benchmark improves**, lifting the overall average by **10 percentage points**.

	MMLU	GSM8K	ARC-C	HSwag	HEval	MBPP	WinoG	Avg.
Pretrained	29.8	14.0	48.4	67.3	4.9	0.4	59.6	32.1
Mid-trained	**46.4**	**44.4**	**52.3**	**70.3**	**8.5**	**11.0**	**64.2**	**42.4**

Table 15. Steerling-8B before and after mid-training across the LM Harness suite (accuracy, %). HSwag: HellaSwag; HEval: HumanEval; WinoG: WinoGrande.

	Pretrained	Mid-trained
Concept Loss ↓	0.002	0.002
Concept Independence Loss ↓	1.907	**1.546**
Concept Contribution ↑	0.851	**0.876**
Known Concept Alignment ↑	3.730	**3.770**

Table 16. Interpretability metrics for Steerling-8B before and after mid-training.

Mid-training improves interpretability. The effects of midtraining on interpretability metrics are shown in Table 16. The late-pretraining rise in concept independence loss, the third weakness of Section 9.4, is mitigated after mid-training: the two-term penalty of Section 10.2 brings concept independence loss down by 19%, so the known and unknown representations are more cleanly separated than at the end of pretraining. Concept contribution rises, meaning the concept module accounts for a larger share of each prediction, and known concept alignment improves slightly.

Mid-training improves steering. We evaluate the midtrained Steerling-8B using the steering benchmark used for the pretraining checkpoint (Wu et al., 2025). Midtraining improves every metric in Table 17: mean concept rises from 1.072 to 1.247, mean quality from 0.972 to 1.064, and their harmonic mean from 1.020 to 1.148. We additionally report *mean sample harmonic* (the harmonic mean computed per sample, then averaged across samples). This rises from 0.843 to 0.963.

Checkpoint	Concept ↑	Quality ↑	Harmonic ↑	Sample harmonic ↑
Pretrained (1.2T)	1.072	0.972	1.020	0.843
Mid-trained (1.35T)	**1.247**	**1.064**	**1.148**	**0.963**

Table 17. Steering benchmark scores for the pretrained and mid-trained Steerling-8B checkpoints. Midtraining improves every steering metric.

10.4 Evaluation and results

Table 18 compares Steerling-8B against open base models of comparable size on seven benchmarks, and Figure 31 places those scores on a compute axis. Every peer was trained on **2 to 16**× more compute than Steerling-8B, yet it lands within ∼**10%** of them on average and ahead of the models trained at comparable budgets. Although Steerling-8B is an inherently-interpretable architecture, we find this does not come at the cost of capability.

A model can be both interpretable and competitively performant.

Model	HSwag	WinoG	PIQA	MMLU	ARC-C	GSM8K	Math	Avg.
Steerling-8B	70.3	64.2	75.9	46.4	52.3	44.4	8.0	51.6
LLaMA2 7B	76.0	72.5	79.1	45.9	46.3	13.1	4.3	48.2
DeepSeek 7B	75.4	70.5	79.2	48.2	48.1	17.4	6.0	49.3
Gemma 1 7B	81.2	72.3	81.2	64.3	53.2	46.4	24.3	60.4
LLaDA 8B	70.5	74.8	73.6	65.9	45.9	70.3	31.4	61.8
LLaMA3 8B	79.1	77.3	80.6	65.4	53.1	48.7	16.0	60.0
OLMo 2 7B	83.8	77.2	80.1*	63.7	79.8	67.5	19.1*	67.3

Table 18. Steerling-8B base model against open base models of comparable size. HSwag: HellaSwag; WinoG: WinoGrande. Values marked * are taken from the OLMo 3 report (OLMo et al., 2025); all other peer numbers are from the respective model reports.

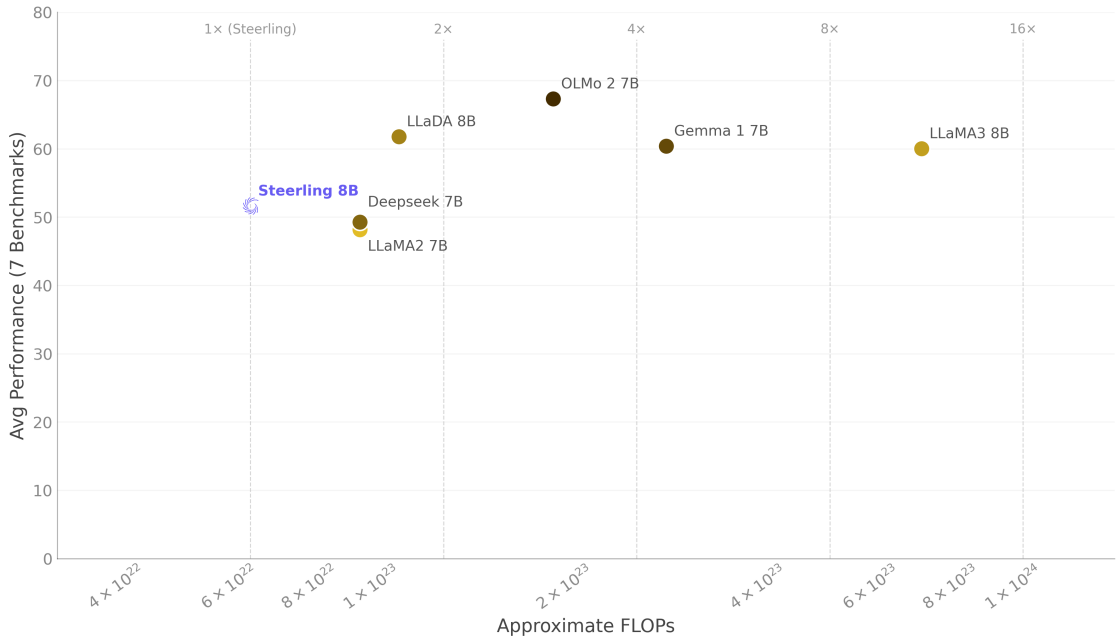

Figure 31. Average performance across the benchmark suite against approximate training FLOPs, with vertical lines at multiples of Steerling-8B's compute.

11 Related work

11.1 Underspecification and the Rashomon effect

The phenomenon. For overparameterized neural networks, many distinct models achieve equivalent performance on the training objective while differing in their internal mechanisms (D'Amour et al., 2022; Breiman, 2001; Black et al., 2022; Rudin et al., 2024). Fisher et al. (2018) formalized this as the *Rashomon set*, the collection of models within a small tolerance of the optimum, and showed that feature importance varies substantially across its members. Semenova et al. (2022) demonstrated that these sets are large in practice, with structurally diverse models routinely coexisting at equivalent performance.

The problem for post-hoc interpretability. Even a perfectly faithful explanation is an account of one arbitrary member of the Rashomon set; a different training run could yield an equally valid model with an entirely different internal decomposition. Worse, Brunet et al. (2022) show that models with nearly identical accuracy can produce contradicting explanations, e.g. opposite-sign attributions for the same feature, with no diagnostic to predict when this occurs. Pawelczyk et al. (2020) show that counterfactual recommendations derived from one model's decision boundary can be invalid for an equally valid alternative.

How inherent interpretability addresses this? Our approach constrains the Rashomon set during training. The concept library is fixed before training begins, the concept module forces every trained model to decompose its output through the same concept variables, and the masking objective ensures a shared absence baseline. Different runs still yield different parameters, but the attribution interface: which concepts exist, how attribution is computed, and what "absent" means, is fixed.

11.2 Large language models

Most large language models are autoregressive (AR): they generate a sequence left to right, one token at a time, each conditioned on all preceding tokens (Radford et al., 2018). Trained at scale with a next-token prediction objective, these models attain strong performance across a wide range of tasks, with training recipes and open model weights now widely documented by the community (Grattafiori et al., 2024b; OLMo et al., 2025; Liu et al., 2024). Most deployed production models belong to this family (Team et al., 2023; Anthropic, 2024; Achiam et al., 2023), making the AR pipeline a mature and well-understood baseline.

Diffusion language models generate by iteratively denoising a corrupted sequence, producing many tokens in parallel and in arbitrary order. Discrete diffusion comes in several families that trade off quality and efficiency: masked diffusion attains the strongest perplexity (Nie et al., 2024, 2025), uniform-state diffusion yields higher-quality samples in the few-step regime and is well suited to guidance (Austin et al., 2021b; Schiff et al., 2025; Sahoo et al., 2025a), and interpolating diffusion supports KV caching for faster decoding (Sahoo et al., 2025b). Block Diffusion bridges the two paradigms by factorizing autoregressively over blocks while denoising within each block, recovering KV caching at the cost of full block-wise attention (Arriola et al., 2025). Recent work has further shown that masked diffusion models scale to billions of parameters (Nie et al., 2025) and now serve production workloads (Song et al., 2025; Labs et al., 2025). Masked diffusion models are also a natural substrate for interpretability: the `[MASK]` token gives a learned baseline for attribution, parallel generation supports concept-level control, and any-order generation enables clean interventions. Steerling builds on this with Causal Diffusion, a block-causal formulation that attains the benefits of Block Diffusion at roughly half the cost, and adds the concept module on top.

11.3 Scaling laws

Scaling laws characterize how model performance improves with compute, parameters, and data, and have become the standard tool for planning large training runs. For autoregressive language models, this line of work established power-law relationships between loss and scale and the compute-optimal allocation of parameters and tokens (Kaplan et al., 2020; Hoffmann et al., 2022; Bi et al., 2024). More recently, the same methodology has been extended to diffusion language models, both for masked diffusion (Nie et al., 2024; von Rütte et al., 2025) and through compute-optimal studies tailored to the diffusion objective (Ni et al., 2025; Sahoo et al., 2026). We follow this methodology and fit compute-optimal scaling laws to inherently interpretable models, measuring the effect of the concept module on both the autoregressive and diffusion families.

A separate line of work asks whether interpretability itself scales. Sparse autoencoders trained on frozen model activations recover more and finer features as the autoencoder grows, with reconstruction quality and feature-quality metrics following clean power laws (Gao et al., 2025; Templeton, 2024), and recurring neuron populations become more selective and monosemantic as the base model grows (Dravid et al., 2026). These works ask whether a post-hoc probe gets better as the probe is scaled, on a fixed, uninterpretable model. They do not ask how a model's own interpretability scales with its training compute. Adapting the irreducible-loss scaling form of Gao et al. (2025), we test whether interpretability-by-design preserves compute-optimal scaling and whether the model's interpretability metrics improve predictably with compute across several distinct measures.

11.4 Interpretable-by-design architectures

A range of architectures build interpretability into the model rather than recovering it post hoc. Among these, Concept Bottleneck Models (CBMs) (Koh et al., 2020) route predictions through a layer of human-understandable concepts. Concept Embedding Models relax the bottleneck to recover accuracy (Espinosa Zarlenga et al., 2022). Originally developed for classification, the approach was later extended to generative models, where intervening on the concept layer enables interpretable and controllable generation (Ismail et al., 2024). It has since reached protein language modeling (Ismail et al., 2025) and single-cell counterfactuals generation (Andersson et al., 2026). Backpack language models pursue a similar goal through a different mechanism, attaching interpretable sense vectors to each token in place of named concepts (Hewitt et al., 2023). All of these operate at a substantially smaller scale, and with far fewer concepts, than Steerling. We use an additive CBM to scale the concept vocabulary to over a *hundred thousand concepts*, with the goal of reaching millions.

11.5 Attribution methods

Input attribution. We use Integrated Gradients (Sundararajan et al., 2017a), but the core of our input attribution is not the algorithm: it is the choice of baseline. Prior Integrated Gradient implementations use a zero or padding embedding as the baseline (Kokhlikyan et al., 2020; Nguyen et al., 2021); the model never learned to interpret either as absence of information. We use `[MASK]` instead. The diffusion training objective makes `[MASK]` a learned representation of "no information at this position," so the integration path stays inside the model's training distribution. Autoregressive models offer no comparable learned baseline through their training objective.

Concept attribution. Methods that attribute predictions to human-interpretable concepts divide into post-hoc and architectural approaches. Post-hoc methods include linear probes (Alain and Bengio, 2016) and TCAV (Kim et al., 2018), which detect concepts in activations or test their influence on predictions, and sparse autoencoders (Huben et al., 2024; Gao et al., 2025), which factorize activations into interpretable features. All are approximations of an internal representation that was not designed to be decomposed. Architectural methods embed concept supervision in the model itself, and we cover this family in Section 11.4. Steerling sits in the architectural family and produces an exact additive decomposition of every output logit, so concept attribution reads off the forward pass rather than estimating it.

Training data attribution. Influence functions estimate the effect of perturbing a training point on a model's predictions (Koh and Liang, 2017). Scaling them to large language models requires approximations to the Hessian inverse (Grosse et al., 2023), and the resulting estimates can be brittle in non-convex regimes (Bae et al., 2022). TracIn takes a different approach, tracing influence along the optimization trajectory (Pruthi et al., 2020). A complementary line avoids estimation altogether: OLMoTrace (Liu et al., 2025) retrieves verbatim substring matches between an output and the training corpus, surfacing documents the model could have seen. We adopt the same retrieval framing but match in latent representation space rather than by surface form: given an output, we return the training chunks most similar to it in that space. We do not claim our method estimates causal influence at trillion-token scale.

11.6 Model steering

Methods for steering model behavior at inference time fall into two families. Both add a learned direction to hidden activations, with the sign of the injection determining whether the target behavior is amplified or suppressed. They differ in how the direction is obtained. The first family derives

the direction from positive and negative examples of the target behavior, generating a new direction per task: representation engineering (Zou et al., 2023), steering vectors (Turner et al., 2023), and contrastive activation addition (Rimsky et al., 2024). The second family selects directions from a pre-built feature dictionary that is independent of any specific behavior we desire to steer. Parsimonious concept engineering (Luo et al., 2024) and sparse autoencoder features (Huben et al., 2024) decompose activations into a fixed set of interpretable features, any of which can serve as a steering direction. Both families obtain the direction *post-hoc*, from activations the model has already produced. Our model instead has its steering directions built into the architecture: each concept contributes an exact additive term to every output logit, so positive or negative steering is a closed-form edit on that term, requiring neither contrastive derivation nor post-hoc dictionary construction.

12 Conclusion

Interpretability is often treated as a cost paid against capability. This paper tested whether that cost grows with scale. We find that it does not. Across three orders of magnitude of compute, on both autoregressive and causal-diffusion language models, adding interpretable structure shifts compute-optimal scaling by a small, fixed offset rather than a penalty that compounds with scale. More surprisingly, interpretability improves with compute: larger models have more independent, and more semantically aligned use of human-understandable concepts. Under the metrics we measure, the model does not become harder to understand as it becomes more capable; it becomes easier.

The central change is where interpretability enters the modeling process. Standard pipelines train opaque predictors and then ask whether post-hoc methods can recover faithful explanations. Instead, we ask which conditions training must enforce for explanations to be faithful, and build those conditions into the data, architecture, objective, and losses. The resulting recipe is not a collection of interpretability add-ons: each component exists because removing it breaks a specific condition required for faithful attribution. Because the same concept variables support both attribution and intervention, a user can decompose an output, inspect the relevant concepts and similar training data, edit the responsible concept direction, and verify the immediate logit-level effect, all without retraining.

The choices made in this work are first instantiations, not settled directions. We fixed the concept library before training, to topics mostly describing the content of training document segments. We chose one bottleneck design, the additive module, among many possible alternatives. We supervised concepts at the chunk level, and our training stops at supervised finetuning. Each of these choices can be improved upon, and we expect future work to do so substantially.

A few prospects seem tantalizing. The first is post-training: reward objectives can be expressed over concepts, so that training targets not only what a model says, but which concepts it uses to decide. In agentic settings, the same structure lets an agent's decisions be decomposed into inspectable concepts that are monitored and corrected mid-trajectory. Concept libraries can grow to match, becoming hierarchical and adaptive rather than fixed, with domains. Furthermore, individuals should be able to interactively define, audit, and extend the vocabularies through which models explain themselves. Second, the attribution interfaces can become more legible in turn: today they return structured artifacts, ranked concept contributions, token-level scores, and retrieved training chunks; future work can translate these into natural-language explanations that remain grounded in the underlying decomposition.

Third, our scaling laws suggest that a frontier-grade interpretable model is feasible, including agentic

systems whose every action can be decomposed, audited, and steered. We therefore view Steerling-8B less as a finished system, but as evidence that this research program is viable.

Taken together, the results suggest a different scaling paradigm for capable AI systems. Interpretability, steerability, and other reliability requirements need not be retrofitted after training, nor treated as a tax against capability. They can be specified as a contract, optimized as part of the training process, and measured as models scale. Our results point to another possibility: if interpretability can be specified, trained, and scaled like any other capability, then the opacity of today's most capable systems is not a law of nature.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone, 2024. *URL https://arxiv. org/abs/2404.14219*, 2 (6):4, 2024.

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.

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

Alma Andersson, Aya Abdelsalam Ismail, Edward De Brouwer, Doron Haviv, Tommaso Biancalani, Kyunghyun Cho, Gabriele Scalia, Aicha BenTaieb, and Hector Corrada Bravo. scCBGM: Single-cell editing via concept bottlenecks. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Anthropic. The Claude 3 model family: Opus, Sonnet, Haiku, 2024. URL https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf.

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*, 2025.

Pepa Atanasova, Oana-Maria Camburu, Christina Lioma, Thomas Lukasiewicz, Jakob Grue Simonsen, and Isabelle Augenstein. Faithfulness tests for natural language explanations. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 283–294, 2023.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021a.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021b.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics, 2023a.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*, 2023b.

Juhan Bae, Nathan Ng, Alston Lo, Marzyeh Ghassemi, and Roger B Grosse. If influence functions are the answer, then what is the question? *Advances in Neural Information Processing Systems*, 35: 17953–17967, 2022.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Pietro Barbiero, Giovanni De Felice, Mateo Espinosa Zarlenga, Francesco Giannini, Filippo Bonchi, Mateja Jamnik, Giuseppe Marra, and Ruggero Noris. The standard interpretable model: A general theory of interpretable machine learning to deductively design interpretable methods using lagrangian mechanics, 2026.

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

Lukas Berglund, Meg Tong, Maximilian Kaufmann, Mikita Balesni, Asa Stickland, Tomek Korbak, and Owain Evans. The reversal curse: Llms trained on "a is b" fail to learn "b is a". In *International Conference on Learning Representations*, volume 2024, pages 18623–18642, 2024.

Usha Bhalla, Suraj Srinivas, Asma Ghandeharioun, and Himabindu Lakkaraju. Towards unifying interpretability and control: Evaluation via intervention. *arXiv preprint arXiv:2411.04430*, 2024.

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

Emily Black, Manish Raghavan, and Solon Barocas. Model multiplicity: Opportunities, concerns, and solutions. In *Proceedings of the 2022 ACM conference on fairness, accountability, and transparency*, pages 850–863, 2022.

Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008 (10):P10008, 2008.

Leo Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231, 2001.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, et al. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2(5):6, 2023a.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, et al. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2(5):6, 2023b.

Marc-Etienne Brunet, Ashton Anderson, and Richard Zemel. Implications of model indeterminacy for explanations of automated decisions. *Advances in Neural Information Processing Systems*, 35: 7810–7823, 2022.

Yanda Chen, Joe Benton, Ansh Radhakrishnan, Jonathan Uesato, Carson Denison, John Schulman, Arushi Somani, Peter Hase, Misha Wagner, Fabien Roger, et al. Reasoning models don't always say what they think. *arXiv preprint arXiv:2505.05410*, 2025.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems, 2021. *URL https://arxiv. org/abs/2110.14168*, 9, 2021.

Alexis Conneau, German Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. What you can cram into a single $ &!#* vector: Probing sentence embeddings for linguistic properties. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2126–2136, 2018.

Alexander D'Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D Hoffman, et al. Underspecification presents challenges for credibility in modern machine learning. *Journal of Machine Learning Research*, 23(226):1–61, 2022.

David L Davies and Donald W Bouldin. A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, PAMI-1(2):224–227, 1979.

Francesco De Comité, François Denis, Rémi Gilleron, and Fabien Letouzey. Positive and unlabeled examples help learning. In *International conference on algorithmic learning theory*, pages 219–230. Springer, 1999.

François Denis. Pac learning from positive statistical queries. In *International conference on algorithmic learning theory*, pages 112–126. Springer, 1998.

François Denis, Rémi Gilleron, and Fabien Letouzey. Learning from positive and unlabeled examples. *Theoretical Computer Science*, 348(1):70–83, 2005.

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*, pages 4443–4458, 2020.

Ann-Kathrin Dombrowski, Maximillian Alber, Christopher Anders, Marcel Ackermann, Klaus-Robert Müller, and Pan Kessel. Explanations can be manipulated and geometry is to blame. *Advances in neural information processing systems*, 32, 2019.

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

Amil Dravid, Yasaman Bahri, Alexei A. Efros, and Yossi Gandelsman. Neuron populations exhibit divergent selectivity with scale, 2026. URL https://arxiv.org/abs/2606.03990.

Yanai Elazar, Shauli Ravfogel, Alon Jacovi, and Yoav Goldberg. Amnesic probing: Behavioral explanation with amnesic counterfactuals. *Transactions of the Association for Computational Linguistics*, 9:160–175, 2021.

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.

Mateo Espinosa Zarlenga, Pietro Barbiero, Gabriele Ciravegna, Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, Zohreh Shams, Frederic Precioso, Stefano Melacci, Adrian Weller, et al. Concept embedding models: Beyond the accuracy-explainability trade-off. *Advances in neural information processing systems*, 35:21400–21413, 2022.

Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong but many are useful: variable importance for black-box, proprietary, or misspecified prediction models, using model class reliance. *arXiv preprint arXiv:1801.01489*, 1, 2018.

W. Nelson Francis and Henry Kučera. *Computational Analysis of Present-Day American English*. Brown University Press, 1967.

Leo Gao, Tom Dupre la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. In *International Conference on Learning Representations*, volume 2025, pages 26721–26754, 2025.

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*, 2024a.

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*, 2024b.

Daniel Greenfeld and Uri Shalit. Robust learning with the hilbert-schmidt independence criterion. In *International Conference on Machine Learning*, pages 3759–3768. PMLR, 2020.

Roger Grosse, Juhan Bae, Cem Anil, Nelson Elhage, Alex Tamkin, Amirhossein Tajdini, Benoit Steiner, Dustin Li, Esin Durmus, Ethan Perez, et al. Studying large language model generalization with influence functions. *arXiv preprint arXiv:2308.03296*, 2023.

Will Held. Scaling laws that extrapolate 300× past the fit. https://www.openathena.ai/blog/delphi/, may 2026. Open Athena Blog.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

John Hewitt and Christopher D Manning. A structural probe for finding syntax in word representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4129–4138, 2019.

John Hewitt, John Thickstun, Christopher D Manning, and Percy Liang. Backpack language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9103–9125, 2023.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, DDL Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 10, 2022.

Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. A benchmark for interpretability methods in deep neural networks. *Advances in neural information processing systems*, 32, 2019.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

Robert Huben, Hoagy Cunningham, Logan Smith, Aidan Ewart, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *International Conference on Learning Representations*, volume 2024, pages 7827–7845, 2024.

Aya Abdelsalam Ismail, Julius Adebayo, Hector Corrada Bravo, Stephen Ra, and Kyunghyun Cho. Concept bottleneck generative models. In *International Conference on Learning Representations*, volume 2024, pages 34114–34132, 2024.

Aya Abdelsalam Ismail, Tuomas Oikarinen, Amy Wang, Julius Adebayo, Samuel Stanton, Taylor Joren, Joseph Kleinhenz, Allen Goodman, Héctor Corrada Bravo, Kyunghyun Cho, et al. Concept bottleneck language models for protein design. *The Thirteenth International Conference on Learning Representations*, 2025.

Daniel Mingyi Israel, Aditya Grover, and Guy Van den Broeck. Enabling autoregressive models to fill in masked tokens. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 4954–4965, 2026.

Alon Jacovi and Yoav Goldberg. Towards faithfully interpretable nlp systems: How should we define and evaluate faithfulness? In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 4198–4205, 2020.

Saachi Jain, Hadi Salman, Eric Wong, Pengchuan Zhang, Vibhav Vineet, Sai Vemprala, and Aleksander Madry. Missingness bias in model debugging. *arXiv preprint arXiv:2204.08945*, 2022.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE transactions on big data*, 7(3):535–547, 2019.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, et al. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International conference on machine learning*, pages 2668–2677. PMLR, 2018.

Ryuichi Kiryo, Gang Niu, Marthinus C Du Plessis, and Masashi Sugiyama. Positive-unlabeled learning with non-negative risk estimator. *Advances in neural information processing systems*, 30, 2017.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR, 2017.

Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *International conference on machine learning*, pages 5338–5348. PMLR, 2020.

Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, et al. Captum: A unified and generic model interpretability library for pytorch. *arXiv preprint arXiv:2009.07896*, 2020.

Tomek Korbak, Mikita Balesni, Elizabeth Barnes, Yoshua Bengio, Joe Benton, Joseph Bloom, Mark Chen, Alan Cooney, Allan Dafoe, Anca Dragan, et al. Chain of thought monitorability: A new and fragile opportunity for ai safety. *arXiv preprint arXiv:2507.11473*, 2025.

Anton Korznikov, Andrey Galichin, Alexey Dontsov, Oleg Y. Rogov, Ivan Oseledets, and Elena Tutubalina. Sanity checks for sparse autoencoders: Do SAEs beat random baselines? *arXiv preprint arXiv:2602.14111*, 2026.

I Elizabeth Kumar, Suresh Venkatasubramanian, Carlos Scheidegger, and Sorelle Friedler. Problems with shapley-value-based explanations as feature importance measures. In *International conference on machine learning*, pages 5491–5500. PMLR, 2020.

Inception Labs, Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, et al. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*, 2025.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, et al. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*, 2023.

Patrick Leask, Bart Bussmann, Michael Pearce, Joseph Bloom, Curt Tigges, Noura Al Moubayed, Lee Sharkey, and Neel Nanda. Sparse autoencoders do not find canonical units of analysis. In *ICLR*, 2025. arXiv:2502.04878.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, et al. Datacomp-lm: In search of the next generation of training sets for language models. *Advances in Neural Information Processing Systems*, 37:14200–14282, 2024.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

Library of Congress. Library of Congress Classification. https://www.loc.gov/catdir/cpso/lcc.html, 2023. Accessed 2026-06-04.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Jiacheng Liu, Taylor Blanton, Yanai Elazar, Sewon Min, Yen-Sung Chen, Arnavi Chheda-Kothary, Huy Tran, Byron Bischoff, Eric Marsh, Michael Schmitz, et al. Olmotrace: Tracing language model outputs back to trillions of training tokens. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 178–188, 2025.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. The flan collection: Designing data and methods for effective instruction tuning. In *International conference on machine learning*, pages 22631–22648. PMLR, 2023.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

Jinqi Luo, Tianjiao Ding, Kwan H Chan, Darshan Thaker, Aditya Chattopadhyay, Chris Callison-Burch, and René Vidal. Pace: Parsimonious concept engineering for large language models. *Advances in Neural Information Processing Systems*, 37:99347–99381, 2024.

Qing Lyu, Marianna Apidianaki, and Chris Callison-Burch. Towards faithful model explanation in nlp: A survey. *Computational Linguistics*, 50(2):657–723, 2024.

Andreas Madsen, Sarath Chandar, and Siva Reddy. Are self-explanations from large language models faithful? In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 295–337, 2024a.

Andreas Madsen, Himabindu Lakkaraju, Siva Reddy, and Sarath Chandar. Interpretability needs a new paradigm. *arXiv preprint arXiv:2405.05386*, 2024b.

Anita Mahinpei, Justin Clark, Isaac Lage, Finale Doshi-Velez, and Weiwei Pan. Promises and pitfalls of black-box concept learning models. *arXiv preprint arXiv:2106.13314*, 2021.

Rowan Hall Maudslay and Ryan Cotterell. Do syntactic probes probe syntax? experiments with jabberwocky probing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 124–131, 2021.

Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Maxime Méloux, Giada Dirupo, François Portet, and Maxime Peyrard. The dead salmons of ai interpretability. *arXiv preprint arXiv:2512.18792*, 2025.

Microsoft. Blingfire tokenizer. https://github.com/microsoft/BlingFire, 2019.

Mistral AI Team. Mistral small 24b instruct (version 2501), 2025. URL https://huggingface.co/mistralai/Mistral-Small-24B-Instruct-2501. Accessed: 2026-06-04.

Joris Mooij, Dominik Janzing, Jonas Peters, and Bernhard Schölkopf. Regression by dependence minimization and its application to causal inference in additive noise models. In *Proceedings of the 26th annual international conference on machine learning*, pages 745–752, 2009.

Giang Nguyen, Daeyoung Kim, and Anh Nguyen. The effectiveness of feature attribution methods and its correlation with automatic evaluation scores. *Advances in Neural Information Processing Systems*, 34:26422–26436, 2021.

Jinjie Ni, Qian Liu, Chao Du, Longxu Dou, Hang Yan, Zili Wang, Tianyu Pang, and Michael Qizhe Shieh. Training optimal large diffusion language models. *arXiv preprint arXiv:2510.03280*, 2025.

Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. Scaling up masked diffusion models on text. *arXiv preprint arXiv:2410.18514*, 2024.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, JUN ZHOU, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

Team OLMo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. *arXiv preprint arXiv:2406.03736*, 2024.

Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothesis and the geometry of large language models. *arXiv preprint arXiv:2311.03658*, 2023.

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. Openwebmath: An open dataset of high-quality mathematical web text. In *International Conference on Learning Representations*, volume 2024, pages 20357–20379, 2024a.

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. Openwebmath: An open dataset of high-quality mathematical web text. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=jKHmjlpViu.

Martin Pawelczyk, Klaus Broelemann, and Gjergji Kasneci. On counterfactual explanations under predictive multiplicity. In *Conference on Uncertainty in Artificial Intelligence*, pages 809–818. PMLR, 2020.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33: 19920–19930, 2020.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018.

Abhilasha Ravichander, Yonatan Belinkov, and Eduard Hovy. Probing the probing paradigm: Does probing accuracy entail task relevance? In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3363–3377, 2021.

Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Turner. Steering llama 2 via contrastive activation addition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15504–15522, 2024.

Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

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.

Cynthia Rudin, Chudi Zhong, Lesia Semenova, Margo Seltzer, Ronald Parr, Jiachang Liu, Srikar Katta, Jon Donnelly, Harry Chen, and Zachery Boner. Amazing things come from having many good models. *arXiv preprint arXiv:2407.04846*, 2024.

Subham S Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin Chiu, and Volodymyr Kuleshov. The diffusion duality. *arXiv preprint arXiv:2506.10892*, 2025a.

Subham Sekhar Sahoo, Zhihan Yang, Yash Akhauri, Johnna Liu, Deepansha Singh, Zhoujun Cheng, Zhengzhong Liu, Eric Xing, John Thickstun, and Arash Vahdat. Esoteric language models. *arXiv preprint arXiv:2506.01928*, 2025b.

Subham Sekhar Sahoo, Jean-Marie Lemercier, Zhihan Yang, Justin Deschenaux, Jingyu Liu, John Thickstun, and Ante Jukic. Scaling beyond masked diffusion language models. *arXiv preprint arXiv:2602.15014*, 2026.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Yair Schiff, Subham Sahoo, Hao Phung, Guanghan Wang, Sam Boshar, Hugo Dalla-Torre, Bernardo Almeida, Alexander Rush, Thomas Pierrot, and Volodymyr Kuleshov. Simple guidance mechanisms for discrete diffusion models. In *International Conference on Learning Representations*, volume 2025, pages 43776–43821, 2025.

Ramprasaath R Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? *arXiv preprint arXiv:1611.07450*, 2016.

Lesia Semenova, Cynthia Rudin, and Ronald Parr. On the existence of simpler machine learning models. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 1827–1858, 2022.

Sentence Transformers Team. all-mpnet-base-v2 model card, 2021. URL https://huggingface.co/sentence-transformers/all-mpnet-base-v2. Accessed: 2026-06-04.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37:103131–103167, 2024.

Xian Shuai, Yiding Wang, Yimeng Wu, Xin Jiang, and Xiaozhe Ren. Scaling law for language models training considering batch size. *arXiv preprint arXiv:2412.01505*, 2024.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Luca Soldaini and Kyle Lo. peS2o (Pretraining Efficiently on S2ORC) Dataset. Technical report, Allen Institute for AI, 2023. ODC-By, https://github.com/allenai/pes2o.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, et al. Dolma: An open corpus of three trillion tokens for language model pretraining research. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15725–15788, 2024.

Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, et al. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

Suraj Srinivas, Kyle Matoba, Himabindu Lakkaraju, and François Fleuret. Efficient training of low-curvature neural networks. *Advances in Neural Information Processing Systems*, 35:25951–25964, 2022.

Suraj Srinivas, Sebastian Bordt, and Himabindu Lakkaraju. Which models have perceptually-aligned gradients? an explanation via off-manifold robustness. *Advances in neural information processing systems*, 36:21172–21195, 2023.

Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-cc: Transforming common crawl into a refined long-horizon pretraining dataset. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2459–2475, 2025.

Nishant Subramani, Nivedita Suresh, and Matthew E Peters. Extracting latent steering vectors from pretrained language models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 566–581, 2022.

Chung-En Sun, Tuomas Oikarinen, Berk Ustun, and Tsui-Wei Weng. Concept bottleneck large language models. *The Thirteenth International Conference on Learning Representations*, 2025.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017a.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017b.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Michal Guerquin, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 OLMo 2 Furious, 2024. URL https://arxiv.org/abs/2501.00656.

Adly Templeton. *Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet*. Anthropic, 2024.

Yury Tokpanov, Paolo Glorioso, Quentin Anthony, and Beren Millidge. Zyda-2: a 5 trillion token high-quality dataset. *arXiv preprint arXiv:2411.06068*, 2024.

Alan Tseng. Library Classification Systems. https://huggingface.co/datasets/agentlans/library-classification-systems, 2024. Hugging Face dataset; accessed 2026-06-04.

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

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Elena Voita and Ivan Titov. Information-theoretic probing with minimum description length. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 183–196, 2020.

Dimitri von Rütte, Janis Fluri, Omead Pooladzandi, Bernhard Schölkopf, Thomas Hofmann, and Antonio Orvieto. Scaling behavior of discrete diffusion language models. *arXiv preprint arXiv:2512.10858*, 2025.

Maurice Weber, Daniel Y. Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Ré, Irina Rish, and Ce Zhang. Redpajama: an open dataset for training large language models. *NeurIPS Datasets and Benchmarks Track*, 2024.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022a.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022b.

Zhengxuan Wu, Aryaman Arora, Atticus Geiger, Zheng Wang, Jing Huang, Dan Jurafsky, Christopher D Manning, and Christopher Potts. Axbench: Steering llms? even simple baselines outperform sparse autoencoders. *arXiv preprint arXiv:2501.17148*, 2025.

Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Beyond autoregression: Discrete diffusion for complex reasoning and planning. In *International Conference on Learning Representations*, volume 2025, pages 77875–77898, 2025.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 4791–4800, 2019.

Yanzhao Zhang, M Li, D Long, X Zhang, H Lin, B Yang, P Xie, A Yang, D Liu, J Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models, 2025, 2025.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, et al. Representation engineering: A top-down approach to ai transparency. *arXiv preprint arXiv:2310.01405*, 2023.

Part I

Architecture

A Symbol reference

Symbol	Type	Meaning
Hidden states		
h	vector	Transformer hidden state
$\bar{h}$	vector	Bottlenecked state passed to the LM head
$\bar{h} = \hat{k} + \hat{u} + \varepsilon$	equation	Concept module decomposition
Concept module heads		
f	function	Known head
g	function	Unknown head
$k = \sigma(f(h))$	vector	Per-concept activation probabilities, known
$u = \sigma(g(h))$	vector	Per-concept activation probabilities, unknown
k_{known}	scalar	Top-k count for the known head
k_{unknown}	scalar	Top-k count for the unknown head
Concept embeddings		
K	matrix	Known concept embedding matrix
U	matrix	Unknown concept embedding matrix
K_i	vector	Embedding of known concept i
U_j	vector	Embedding of unknown concept j
n	scalar	Number of known concepts
m	scalar	Number of unknown concepts ($m \gg n$)
R	scalar	Factorization rank of unknown embedding matrix
Concept contributions		
$\hat{k} = \sum_i k_i K_i$	vector	Known concept contribution
$\hat{u} = \sum_j u_j U_j$	vector	Unknown concept contribution
$\varepsilon = h - \hat{k} - \hat{u}$	vector	Residual term
Logit decomposition		
W_y	vector	Row of the LM head for output token y
ℓ_y	scalar	Output logit for token y

Table 19. Concept module notation, grouped by role.

Symbol	Type	Meaning
Losses		
$\mathcal{L}_{\text{LM}}$	loss	Language modeling loss ($\mathcal{L}_{\text{MDM}}$ on $\bar{h}$)
$\mathcal{L}_{\text{concept}}$	loss	Concept loss (chunk-level BCE)
$\mathcal{L}_{\text{rec}}$	loss	Reconstruction loss for the unknown head
$\mathcal{L}_{\text{indep}}$	loss	Independence loss between $\hat{k}$ and $\hat{u}$
$\mathcal{L}$	loss	Combined training objective
$\lambda_{\text{concept}}, \lambda_{\text{rec}}, \lambda_{\text{indep}}$	scalars	Loss weights
Supervision and targets		
y_c	scalar	Ground-truth chunk label, known concept c
k_c^{chunk}	scalar	OR-aggregated chunk-level activation
k_i^{GT}	scalar	Ground-truth activation of known concept i
$\hat{k}^{\text{GT}}$	vector	Ground-truth known concept contribution
$\hat{u}^{\text{GT}} = h - \hat{k}^{\text{GT}}$	vector	Target for the unknown head
Independence loss		
H_k, H_u	matrices	Stacked per-token $\hat{k}$, $\hat{u}$ over a minibatch
$\boldsymbol{\mu}_{\hat{k}}, \boldsymbol{\mu}_{\hat{u}}$	vectors	Column means of H_k, H_u
Φ	matrix	Centered known features
Ψ	matrix	Centered unknown features
Training dynamics		
$\mathcal{M}$	set	Masked token positions in the minibatch
t_b	scalar	Per-block noise level (block b)
$\alpha_{\text{known}}(s)$	scalar	Teacher forcing prob., known, step s
$\alpha_{\text{unknown}}(s)$	scalar	Teacher forcing prob., unknown, step s
p_{cfg}	scalar	Dropout rate for the known head
p_ε	scalar	Dropout rate for the residual ε
B	scalar	Minibatch size
b	scalar	Block size in causal block-diffusion

Table 20. Concept module notation, grouped by role.

Part II

Interpretability capabilities

B Attribution details

B.1 Training data attribution

The training data attribution pipeline of Section 6.1.3 indexes the training corpus offline and retrieves against it at inference time. We describe each component here.

Corpus index. We index the full training corpus, approximately 11 billion chunks, with a fine-tuned embedding model, `FT-Qwen3-Embedding-0.6B`, derived from `Qwen3-Embedding-0.6B` (the annotator model of section 4), encoding each chunk into a 1024-dimensional vector. To make nearest-neighbor search tractable at this scale, we build a FAISS index with an inverted-file structure and product quantization (IVFPQ): the inverted file partitions the corpus into coarse clusters so a query is compared only against the closest few, and product quantization compresses each stored vector to keep the index in memory. Retrieval performs approximate nearest-neighbor search over the top-n coarse clusters.

Chunk representation. At inference time, each chunk of the model response is tokenized and forwarded through the language model to obtain per-token hidden states. These are mean-pooled over the real (non-padding) positions into a single 4096-dimensional representation, formed from the known-head, unknown-head, and residual components of the hidden state, so it encodes the meaning of the chunk as the model represents it.

Transducer. The chunk representation is 4096-dimensional while the corpus index is 1024-dimensional, so we bridge the two spaces with the transducer: a two-hidden-layer MLP with roughly 15 million parameters that maps $\mathbb{R}^{4096} \to \mathbb{R}^{1024}$, trained to preserve semantic content under a cosine-similarity objective,

$$\mathcal{L}_{\text{trans}} = 1 - \cos(\hat{\mathbf{e}}, \mathbf{e}^*)\,, \tag{34}$$

where $\hat{\mathbf{e}}$ is the transducer's predicted embedding and $\mathbf{e}^*$ is the target embedding from `FT-Qwen3-Embedding-0.6B` for the same chunk.

Pipeline. End to end, each response chunk is encoded and mean-pooled into a single representation, transduced into the corpus embedding space, and used to query the IVFPQ index by approximate nearest-neighbor search over the top-n coarse clusters, returning the most similar training chunks as the attributed sources.

Part III

Data

C Atlas: From documents to concepts

Text-assigned concept evaluation. For the tag, concept, and predicted-concept relevance evaluations described in Section 4.2, we use an LLM judge to rate whether candidate concepts are present in a text chunk. The judge is `Mistral-Small-3.1-24B-Instruct` run at temperature 0. It receives a text chunk together with one or more candidate concepts, each represented by a label and one-sentence description, and returns a JSON object assigning each candidate concept a relevance score from 1 to 5. The full prompt is shown in Figure 32.

```
SYSTEM:
You are an expert at evaluating whether concepts are genuinely present in a passage
    of text. For each concept, you need to score based on how well the concept is
    exhibited (either explicitly or in a subtle manner) by the text passage. Use the
     following 1-5 scoring criteria:
- 5: Very strongly present/central to the text - concept is explicitly discussed and
    /or forms a core theme
- 4: Strongly present/well-demonstrated - concept is clearly evident with
    substantial supporting content
- 3: Moderately present/clearly related - concept is reasonably connected with some
    supporting evidence
- 2: Weakly present/tangentially related - minimal connection or only surface-level
    mention
- 1: Not present/completely inappropriate - concept is absent, irrelevant, or text
    merely uses related vocabulary without actually being about that concept

Be precise in your evaluation - consider both explicit mentions and substantive
    demonstration of the concept.

Respond with a single-line JSON object where keys are the concept letters (A, B, C,
    etc.) and values are scores (1-5). **DO NOT include any explanations or
    additional text.**

USER:
Evaluate how well each concept below is exhibited in the given text chunk using the
    1-5 scale.

=== TEXT CHUNK ===
{text_chunk}

=== CONCEPTS TO EVALUATE ===
{concepts_with_descriptions}
```

Figure 32. Prompt for text-assigned concepts evaluation

D Additional details on the human interpretability study

This appendix gives additional methodological and statistical details for the human interpretability study in Section 4.4. The main text reports the study design and headline results; here we report agreement statistics, robustness checks, and model-based analyses.

D.1 Sampling and annotation protocol

We sampled 100 concepts stratified by top-level taxonomy branch: ten concepts from each of the nine largest branches and ten from the aggregated remainder. For each sampled concept, we constructed a lifted-word list by ranking lemmatized words according to

$$\text{lift}(w, c) = \frac{P(w \mid c)}{P(w)},$$

subject to a minimum-support filter. Annotators saw only these lifted words, not the pipeline concept name, concept description, taxonomy position, or source documents.

In Phase 1, annotators wrote a name or short phrase for the concept and rated whether the lifted words formed a recognizable concept on a 1–5 scale. Phase 1 collected 303 named-concept responses from 9 annotators, with a median of 3 annotators per concept. Human-written names averaged 4.1 words.

In Phase 2, annotators rated candidate names for the same lifted-word lists. Each candidate set contained the pipeline's LLM-generated label, two human labels from Phase 1, an embedding-neighbor distractor, and a taxonomy-neighbor distractor. A small number of cases also included a low-effort filler label as a floor control when one of the other candidate types was unavailable. Candidate order was randomized, annotators were blind to candidate provenance, and no annotator scored a label they had written. Phase 2 collected 205 scoring records over 34 concepts from 8 annotators, for $1{,}025$ individual candidate-name ratings. The 20 most-rated concepts were scored by all eight Phase 2 annotators.

D.2 Phase 1 agreement and coherence

Phase 1 was designed to test whether lifted-word evidence contains recoverable semantic structure before any pipeline label is shown. The mean coherence score was 3.52. Annotators judged 55% of responses to form a recognizable concept (≥ 4), rated 27% as borderline ($= 3$), and flagged 17% as incoherent or noisy (≤ 2).

Agreement was moderate rather than perfect, as expected for a task involving short word lists and concepts drawn from many technical domains. ICC(1) was 0.43. The within-concept standard deviation was 0.65, compared with total scale standard deviation 1.05, and 53% of concepts were unanimous on the coherent-vs-not split. These agreement statistics support the main-text conclusion: lifted-word evidence is often meaningful, but not uniformly so.

D.3 Ordinal mixed-model analysis

The primary Phase 2 outcome is an ordinal fit rating $y_i \in \{1, 2, 3, 4, 5\}$ for a candidate label. Because ratings are ordinal and clustered by rater, concept, label, and scoring context, we fit a Bayesian cumulative-link mixed model with crossed random effects. Human labels are the reference category. The model has the form

$$\Pr(y_i \leq k) = \text{logit}^{-1}(\tau_k - \eta_i),$$

Analysis	Result
Mean Phase 2 fit score	LLM 3.98 vs. human 3.50
Top-two rating rate (≥ 4)	LLM 79% vs. human 63%
Bayesian cumulative-link model	OR 2.38, 95% CrI $[1.23, 4.01]$
Posterior probability of LLM advantage	0.99
Paired comparison probability	0.62, bootstrap 95% CI $[0.58, 0.66]$
Gaussian mixed model	$+0.47$ points, 95% CI $[0.28, 0.65]$

Table 21. Robustness checks for the Phase 2 comparison between pipeline labels and independently generated human labels.

where $k \in \{1, 2, 3, 4\}$ indexes the ordinal thresholds and

$$\eta_i = \beta_{\text{LLM}} \mathbf{1}_{\text{LLM},i} + \beta_{\text{emb}} \mathbf{1}_{\text{embedding},i} + \beta_{\text{tax}} \mathbf{1}_{\text{taxonomy},i} + \beta_{\text{fill}} \mathbf{1}_{\text{filler},i} + u^{\text{rater}}_{r[i]} + u^{\text{concept}}_{c[i]} + u^{\text{context}}_{s[i]} + u^{\text{label}}_{\ell[i]}.$$

Positive coefficients indicate higher expected fit scores. The main contrast is β_{LLM}, comparing the pipeline label to human-written labels.

The LLM–human proportional-odds ratio was 2.38 (95% CrI $[1.23, 4.01]$), with posterior probability 0.99 that the pipeline label receives higher fit ratings than a human label. On the response scale, pipeline labels received a top-two rating (≥ 4) 79% of the time, compared with 63% for human labels.

D.4 Assumption-light robustness checks

We also ran simpler checks that make fewer modeling assumptions. In paired comparisons, the pipeline label outscores a human label with probability 0.62 (cluster-bootstrap 95% CI $[0.58, 0.66]$, 402 pairs). A Gaussian mixed model on the raw 1–5 ratings estimates a $+0.47$ point advantage for pipeline labels over human labels (95% CI $[0.28, 0.65]$). These checks agree with the ordinal mixed model: pipeline labels are not merely competitive with human labels, but are rated higher on average under blind evaluation.

D.5 Dependence on lifted-word coherence

If the LLM were hallucinating plausible names uniformly, its advantage should not depend strongly on whether the lifted-word evidence itself is coherent. We therefore examined how Phase 2 label fit varies with Phase 1 coherence.

Concepts with higher Phase 1 coherence receive higher Phase 2 fit scores for both human and pipeline labels. The correlation between Phase 1 coherence and Phase 2 fit is $+0.42$ for human labels and $+0.46$ for pipeline labels. The pipeline advantage is positive across coherence bins, but larger when the underlying word evidence is clearer: the LLM–human gap is $+0.22$ for low-coherence concepts, $+0.54$ for mid-coherence concepts, and $+0.52$ for high-coherence concepts. This pattern is consistent with the LLM naming real statistical structure rather than assigning plausible labels independently of the evidence.

D.6 Power analysis

Finally, we checked whether the realized Phase 2 sample size was sufficient for the observed LLM–human contrast. A simulation under the fitted clustered generative model gives power above 0.99

for the observed contrast at roughly 20 fully-rated concepts and essentially 1.00 at the realized sample size. This analysis should not be interpreted as certifying every individual concept in the full library. Rather, it shows that the stratified pilot is well-powered for the aggregate comparison between pipeline labels and human-written labels.

D.7 Limitations

The human study validates the labeling operation used by Atlas, not every concept individually. A minority of lifted-word lists are ambiguous or noisy, and some concepts require domain expertise that may not be uniformly available across annotators. The Phase 2 comparison also evaluates names relative to lifted-word evidence, rather than full source-document evidence. These limitations make the result conservative in one respect and incomplete in another: humans often recover and endorse the same semantic structure from sparse evidence alone, but the study does not eliminate the need for additional per-domain or per-concept audits in downstream use.

Part IV

Interpretability metrics

E Known concept alignment judge

The Known Concept Alignment metric from section 7 relies on an LLM-judge to rate concepts. The judge uses `Mistral-Small-3.1-24B-Instruct` at temperature 0. It receives a concept's human-assigned label, its one-sentence description, and the top-K tokens scored by the concept embedding through the LM head. It returns a single integer rating from 1 to 5. The full prompt is shown in Figure 33.

```
SYSTEM:
You are an expert in neural network interpretability. You are evaluating whether
a concept head in a language model has learned to represent a specific named
concept.

You will be given:
1. The concept's human-assigned label and description
2. The top activated tokens from the concept head

Rate how well the top tokens represent the named concept on a 1-5 scale:
- 5: Tokens strongly and clearly represent the concept. Most tokens are
     directly related.
- 4: Tokens mostly represent the concept with minor noise or tangential items.
- 3: Tokens partially represent the concept. Some relevant tokens but also
     significant off-topic items.
- 2: Tokens weakly relate to the concept. Only a few tokens connect; most are
     unrelated.
- 1: Tokens do not represent the concept at all.

IMPORTANT: A token does not need to be an exact word from the label or
description to count as relevant. Proper nouns, abbreviations, sub-words, and
semantically related terms all count. For example, if the concept is 'academic
publishers', then 'Penguin', 'Wiley', 'ISBN', 'paperback', and 'imprint' are all
relevant even though none appear in the label.

Respond with ONLY one line:
ALIGNMENT: <integer 1-5>

USER:
Concept label: {concept_label}
Concept description: {concept_description}

=== TOP ACTIVATED TOKENS ===
{top_tokens}
```

Figure 33. Prompt for the Known Concept Alignment judge.

Part V

Scaling laws

F Symbol and notations

Symbol	Type	Meaning
Compute and resources		
C	scalar	Total training FLOPs
M	scalar	Per-token FLOPs (forward + backward)
P	scalar	Non-embedding parameter count
D	scalar	Number of training tokens (total)
D_i	scalar	Training tokens for the i-th checkpoint
Validation losses		
$\mathcal{L}_i$	scalar	Measured validation loss for checkpoint i
$\mathcal{L}^*(C)$	function	Compute-optimal validation loss at budget C
$\mathcal{L}_\infty$	scalar	Irreducible validation loss
$\mathcal{L}(P, D)$	function	Joint Chinchilla loss surface
$\mathcal{L}_{\text{fit}}$	function	Huber loss minimized in the joint fit
Power-law parameters		
$P^*(C)$	function	Compute-optimal parameter count at budget C
a_P, a_L	scalars	Power-law coefficients (parameter and loss)
α_P	scalar	Exponent on compute for parameter scaling
α_L	scalar	Exponent on compute for loss scaling
α_D	scalar	Exponent on compute for training-token scaling
A_P, A_D	scalars	Chinchilla coefficients (parameter and data sides)
α, β	scalars	Chinchilla exponents (parameter and data sides)
Interpretability scaling		
$m(C)$	function	A metric as a function of compute
e	scalar	Irreducible value of a metric
A, β	scalars	Coefficient and exponent for metric scaling

Table 22. Notation introduced in the scaling-law analysis, grouped by role: compute and resources, validation losses, power-law parameters, and interpretability scaling.

G Architectures, IsoFLOP slices, and hyperparameters

Backbones. Table 23 lists the configuration of each backbone size used in the scaling sweep. All four families share the same backbone architecture; +Concept variants add the concept module on top, configured per Table 25 and Table 26.

IsoFLOP slices. Table 24 reports the four IsoFLOP target compute budgets per family. The +Concept families start at higher targets because the concept module's per-token FLOPs are non-negligible at small backbone sizes. Each slice contains four to six model sizes whose per-checkpoint compute lands within $\pm 15\%$ of the target.

Size	Layers L	Hidden d	Backbone params	+Concept total params
10M	6	320	9.2M	82.8M
25M	6	512	23.6M	110.2M
85M	10	768	86.5M	190.5M
200M	13	1024	197.7M	319.1M
400M	17	1280	401.1M	540.0M
800M	17	1792	779.9M	953.6M
1.5B	20	2304	1,510M	1,718M
3B	24	3072	3,228M	3,489M
5B	24	3840	5,694M	6,007M

Table 23. Backbone architectures used across all four families. Backbone parameter counts exclude embeddings; +Concept totals include the concept module heads (concept classifier, and factorized unknown head concept embeddings are excluded). Sequence length is 4096 throughout.

Family	Slice 1	Slice 2	Slice 3	Slice 4
AR	6×10^{18}	10^{19}	3×10^{19}	10^{20}
CDLM	6×10^{18}	10^{19}	3×10^{19}	10^{20}
AR+Concept	10^{19}	3×10^{19}	1.1×10^{20}	3.09×10^{20}
CDLM+Concept	10^{19}	3×10^{19}	1.1×10^{20}	3.09×10^{20}

Table 24. IsoFLOP target compute budgets per family.

Shared hyperparameters. All scaling-law runs share optimizer, learning rate, batch size, warmup, and architectural defaults. The optimizer is AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\varepsilon = 10^{-8}$, weight decay 0.1 (excluding embeddings), and gradient clipping at 1.0. Peak learning rate is fixed at 4×10^{-4} across all model sizes and all families. The schedule is warmup-stable-decay (WSD) with an $80/20$ stable/decay split and decay to zero. Warmup is the minimum of 2000 steps or 2% of total training steps, to accommodate runs with smaller token budgets. The total batch size is $524{,}288$ tokens (128 sequences $\times$ 4096 tokens) across every run. Backbone architectural defaults (post-norm RMSNorm, QK-norm, RoPE base 5×10^5, SwiGLU MLP with ratio 4, no biases, clip_qkv=10, dropout 0) are held fixed across all backbones. Per-(size, slice) token counts are determined by the slice target C and the FLOP equation (Equation (23)); intermediate stable-phase checkpoints are also included in IsoFLOP fits where they fall within $\pm15\%$ of a slice target.

CDLM-specific settings. The two diffusion families (CDLM and CDLM+Concept) share an additional set of settings governing the masking process: causal block size 64, with the noise level sampled uniformly at training time, $t \sim \mathcal{U}(0.05, 0.95)$. These are held fixed across all CDLM model sizes.

Concept module settings. The two +Concept families (AR+Concept, CDLM+Concept) share the concept module configuration. Settings differ between the stable phase (Table 25) and the anneal phase (Table 26), with the anneal phase additionally applying top-k sparsification to both heads. The parameters $\alpha_{\text{known}}(t)$ and $\alpha_{\text{unknown}}(t)$ are the probabilities of using ground-truth components in place of predicted ones during training; see Section 5.4.2 for the motivation and definition.

Setting	Value	Notes
n (known concepts)	33,732	shared across all sizes
Unknown ratio	$3\times$	$\Rightarrow m = 101{,}196$
Factorization rank R	256	for unknown head
Top-k_{known}	16	predictor + compose
$\alpha_{\text{known}}(t)$	$1.0 \to 0.5$ by step $0.10\,T_{\max}$	cosine
$\alpha_{\text{unknown}}(t)$	$0.0 \to 0.5$ by step $0.10\,T_{\max}$	linear
λ_{concept}	1.0	concept loss weight
λ_{rec}	1.0	reconstruction loss weight
λ_{indep}	1.0	independence loss weight
p_{cfg}	0.1	known head dropout rate
p_{ε}	0.3	residual dropout rate

Table 25. Concept module settings during the stable phase. Both α schedules ramp from initial value to 0.5 by step $0.10\,T_{\max}$, then hold constant for the remainder of the stable phase. $T_{\max}$ denotes `max_steps`.

Setting	Value	Notes
$\alpha_{\text{known}}(t)$	$0.5 \to 0.0$ over anneal	linear
$\alpha_{\text{unknown}}(t)$	constant at 1.0	
Top-k_{known}	32	`topk_known`
Top-k_{unknown}	128	factorized compose, `apply_topk_to_unknown`

Table 26. Concept module settings during the anneal phase. The anneal phase resumes from the final stable-phase checkpoint and runs a linear LR decay to zero. Inherited settings (n, unknown ratio, factorization rank, loss weights, dropouts) match Table 25.

H ELBO estimation for validation loss

Autoregressive models report exact negative log-likelihood (cross-entropy on the next token); however, in diffusion models, the per-token NLL cannot be computed exactly and must instead be estimated via Monte Carlo on an Evidence Lower Bound (ELBO). Different MC schemes give materially different absolute loss values, which propagates into reported $\mathcal{L}_\infty$ values and, when the bias is non-uniform across model sizes, into reported scaling exponents. Here we compare four estimators on our CDLM checkpoints. We find that the compute-optimal parameter count $P^*(C)$ and the exponent α_P are robust to estimator choice, while absolute loss values $\mathcal{L}^*$ and the irreducible-loss asymptote $\mathcal{L}_\infty$ vary by up to a nat between estimators.

H.1 Estimators

Let $\mathbf{x}_0 = (x_0^1, \ldots, x_0^N)$ denote a sequence of length N drawn from the validation distribution $\mathcal{D}$. At noise level $t \in [0, 1]$, the forward process replaces each token independently with `[MASK]` with probability t and leaves it unchanged with probability $1 - t$:

$$x_t^i = \begin{cases} \texttt{[MASK]} & \text{with probability } t, \\ x_0^i & \text{with probability } 1 - t. \end{cases} \tag{35}$$

We denote by $\mathcal{M}_t = \{i : x_t^i = \texttt{[MASK]}\}$ the set of masked positions in $\mathbf{x}_t$, and by $p_\theta(x_0^i \mid \mathbf{x}_t)$ the model's predicted distribution over the token at position i given the corrupted sequence $\mathbf{x}_t$. All estimators below average over $\sim$100M tokens of validation data.

Fixed-rate mask loss. The first estimator computes per-token cross-entropy at a single fixed mask rate τ, with no integration over the noise schedule:

$$\hat{\mathcal{L}}_{\text{fixed}}(\tau) = \mathbb{E}_{\mathbf{x}_0,\mathbf{x}_\tau}\left[\frac{1}{|\mathcal{M}_\tau|}\sum_{i\in\mathcal{M}_\tau} -\log p_\theta(x_0^i \mid \mathbf{x}_\tau)\right]. \tag{36}$$

We set $\tau = 0.5$. This estimator computes per-token cross-entropy at one fixed corruption level rather than an ELBO bound on $-\log p_\theta(\mathbf{x}_0)$, but is the closest validation analogue of the training objective, which integrates over $[0.05, 0.95]$.

Fixed-grid discretized ELBO. The second estimator approximates the ELBO integral over t via K fixed bins $\{t_1, \ldots, t_K\}$ averaged uniformly:

$$\hat{\mathcal{L}}_{\text{grid}} = \frac{1}{K}\sum_{k=1}^{K}\mathbb{E}_{\mathbf{x}_0,\mathbf{x}_{t_k}}\left[\frac{1}{|\mathcal{M}_{t_k}|}\sum_{i\in\mathcal{M}_{t_k}} -\log p_\theta(x_0^i \mid \mathbf{x}_{t_k})\right]. \tag{37}$$

We use $K = 9$ with $t_k = k/10$ for $k = 1, \ldots, 9$. With uniform weights this is a midpoint-rule approximation to the ELBO integral over $t \in [0.05, 0.95]$, matching the interval sampled during training.

MDLM ELBO. The third estimator follows the low-variance form of the MDLM ELBO from equation 2. Under the linear schedule $\alpha_t = 1 - t$, the canonical NELBO bound on $-\log p_\theta(\mathbf{x}_0)$ is

$$-\log p_\theta(\mathbf{x}_0) \le \mathbb{E}_{t\sim\mathcal{U}(0,1)}\left[\frac{1}{t}\sum_{i\in\mathcal{M}_t} -\log p_\theta(x_0^i \mid \mathbf{x}_t)\right]. \tag{38}$$

The $1/t$ prefactor cancels in expectation with $\mathbb{E}[|\mathcal{M}_t|/N \mid t] = t$ when $t \sim \mathcal{U}(0,1)$, giving the per-token form

$$\hat{\mathcal{L}}_{\text{MDLM}} = \mathbb{E}_{t\sim\mathcal{U}(0,1)}\,\mathbb{E}_{\mathbf{x}_0,\mathbf{x}_t}\left[\frac{1}{|\mathcal{M}_t|}\sum_{i\in\mathcal{M}_t} -\log p_\theta(x_0^i \mid \mathbf{x}_t)\right]. \tag{39}$$

We sample one t per batch and average over batches. This estimator is unbiased for the per-token NELBO bound and matches the formulations used in Sahoo et al. (2024); Nie et al. (2024); Ni et al. (2025); von Rütte et al. (2025); Sahoo et al. (2026); it is also what we report in section 8.2. Note that the CDLM training loss is itself an unbiased estimator of the same quantity, except that training clips t to $[\beta, \omega] = [0.05, 0.95]$ to avoid degenerate extremes. The two estimators are therefore numerically close but not identical: $\hat{\mathcal{L}}_{\text{MDLM}}$ integrates over the full $[0, 1]$ interval, while the training loss integrates over $[0.05, 0.95]$.

Per-block ELBO. The fourth estimator matches the training distribution of our causal block-diffusion model (Section 5.2), where each block has its own independently sampled noise level. We partition each sequence into N/b blocks of size b. For each block, we sample an independent noise level $t_b \sim \mathcal{U}(0,1)$ and mask positions within that block at rate t_b. We then compute mean cross-entropy on all masked positions:

$$\hat{\mathcal{L}}_{\text{block}} = \mathbb{E}_{\{t_b\}\overset{\text{iid}}{\sim}\mathcal{U}(0,1)}\,\mathbb{E}_{\mathbf{x}_0,\mathbf{x}_t}\left[\frac{1}{|\mathcal{M}_t|}\sum_{i\in\mathcal{M}_t} -\log p_\theta(x_0^i \mid \mathbf{x}_t)\right]. \tag{40}$$

We use $b = 64$, matching the training configuration, so each sequence contributes 64 independent noise levels per validation step. The cancellation argument from Equation (38) applies per-block, so this estimator is also unbiased for the per-token NELBO bound, with lower Monte Carlo variance than $\hat{\mathcal{L}}_{\text{MDLM}}$ because each batch covers a wider distribution of t values.

H.2 Results

IsoFLOP shapes and absolute losses. Figure 34 compares the four estimators across the four CDLM IsoFLOP slices. The per-slice parabolas (top row) have similar shapes and per-slice minima at comparable parameter counts $P^*(C)$ across all estimators, indicating that the location of the compute-optimal model size is robust to estimator choice. The absolute loss values, however, differ substantially: at any fixed slice, the four estimators are vertically offset by up to a nat, with uniform mask consistently lowest and per-block ELBO highest. The bottom row shows each estimator's full IsoFLOP grid; the parabolas tighten with compute under all four estimators, and the slice minima move smoothly toward larger P as C grows.

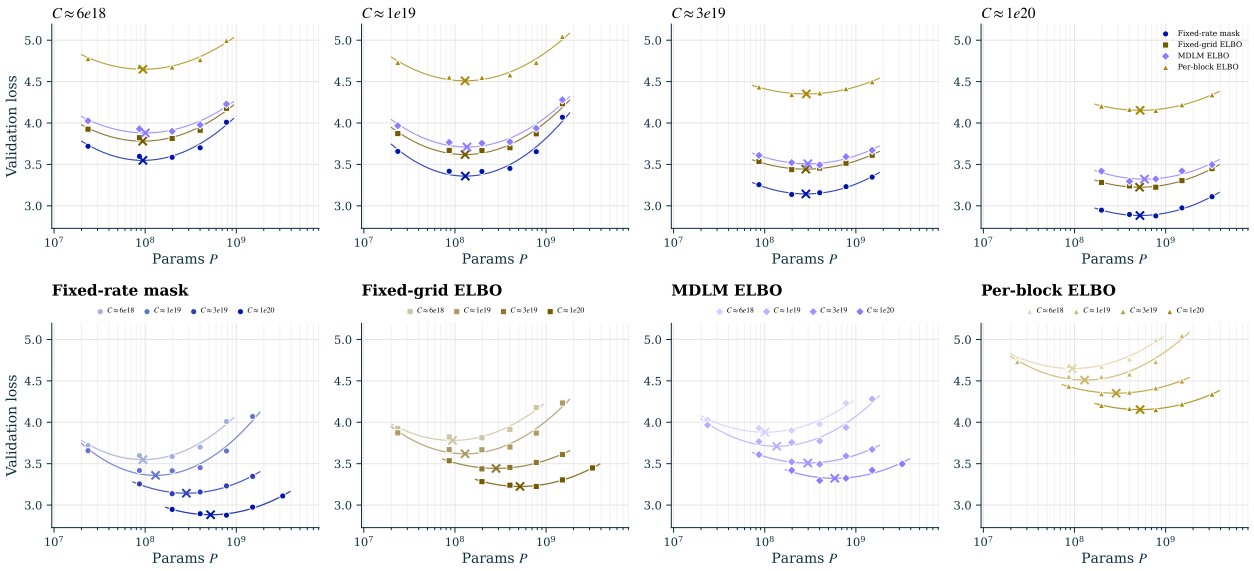

Figure 34. IsoFLOP comparison of the four ELBO estimators on CDLM. Top row: per-slice parabolic fits with all four estimators overlaid. Bottom row: per-estimator IsoFLOP grids with all four slices overlaid.

Effect on power laws. Figure 35 plots the compute-optimal scaling laws $\mathcal{L}^*(C)$ and $P^*(C)$ for each estimator. The $P^*(C)$ fits (panel b) are nearly indistinguishable across estimators, with all four lines overlapping within marker width and α_P values clustering tightly between 0.602 and 0.632. The $\mathcal{L}^*(C)$ fits (panel a) show parallel-ish slopes shifted vertically by an estimator-dependent offset; the loss exponent α_L varies modestly between -0.071 (uniform mask) and -0.039 (per-block ELBO). Table 27 summarizes the exponents with 90% bootstrap confidence intervals.

H.3 Effect of mask rate on scaling exponents

The four estimators above differ in how they aggregate across the noise schedule, mixing the effect of *which* mask rates are sampled with the effect of *how* they are integrated. To isolate the role of the mask rate alone, the IsoFLOP fit is re-run at nine fixed mask rates $t \in \{0.1, 0.2, \ldots, 0.9\}$ on the same CDLM checkpoints. Figure 36 shows the per-slice parabolas at each mask rate, Figure 37

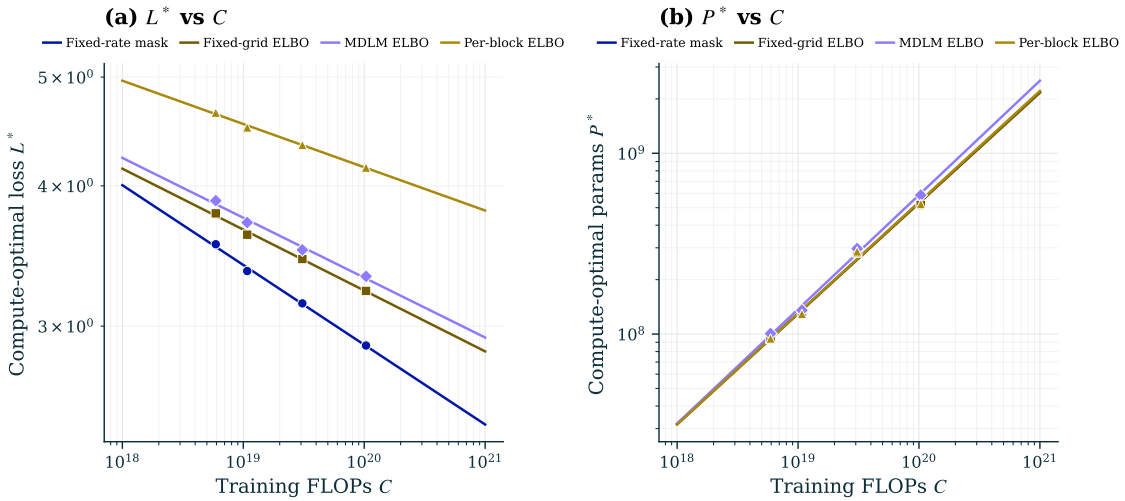

Figure 35. Compute-optimal scaling laws under the four ELBO estimators. (a) $\mathcal{L}^*(C)$ versus compute. (b) $P^*(C)$ versus compute.

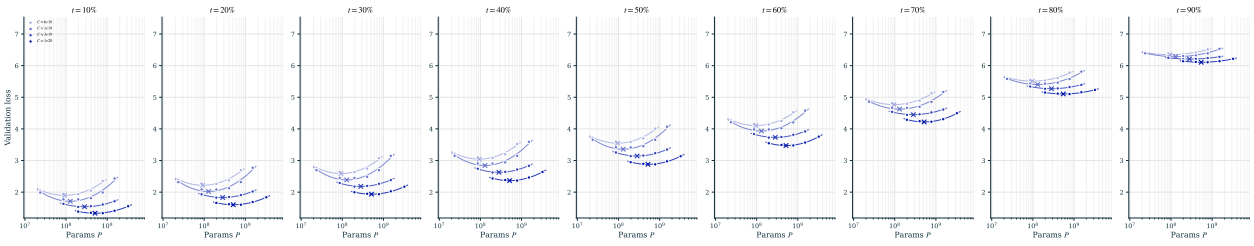

Figure 36. IsoFLOP analysis at nine fixed mask rates $t \in \{0.1, 0.2, \ldots, 0.9\}$ on CDLM. $\times$ markers locate the per-slice minima $P^*(C)$.

Estimator	α_P	α_L	α_D	$\mathcal{L}_\infty$
Fixed-rate mask	0.602 [0.559, 0.650]	−0.071 [−0.075, −0.067]	0.410 [0.339, 0.564]	1.927 [1.447, 2.085]
Fixed-grid ELBO	0.611 [0.568, 0.658]	−0.054 [−0.057, −0.051]	0.368 [0.309, 0.555]	2.416 [1.786, 2.560]
MDLM ELBO	0.632 [0.541, 0.707]	−0.053 [−0.058, −0.049]	0.481 [0.370, 0.643]	2.658 [1.950, 2.839]
Per-block ELBO	0.615 [0.572, 0.660]	−0.039 [−0.041, −0.036]	0.340 [0.281, 0.558]	3.353 [2.685, 3.526]

Table 27. IsoFLOP power-law exponents and irreducible-loss asymptotes under four ELBO estimators on CDLM. Subscripts are 90% bootstrap confidence intervals.

shows the resulting $P^*(C)$ and $\mathcal{L}^*(C)$ fits, and Table 28 reports the exponents.s The parameter exponent α_P separates into two groups: $\alpha_P \approx 0.60$ for $t \leq 0.7$ and $\alpha_P \approx 0.62$ for $t \geq 0.8$. The loss exponent α_L flattens monotonically and substantially, from -0.120 at $t = 0.1$ to -0.013 at $t = 0.9$: at low mask rates the loss has substantial room to fall with compute, while at high mask rates it approaches the random-token floor where additional compute buys little. This decoupling explains the four-estimator pattern documented above: any aggregation across t inherits a similar α_P from the per-rate fits but produces an α_L determined by which mask rates carry weight.

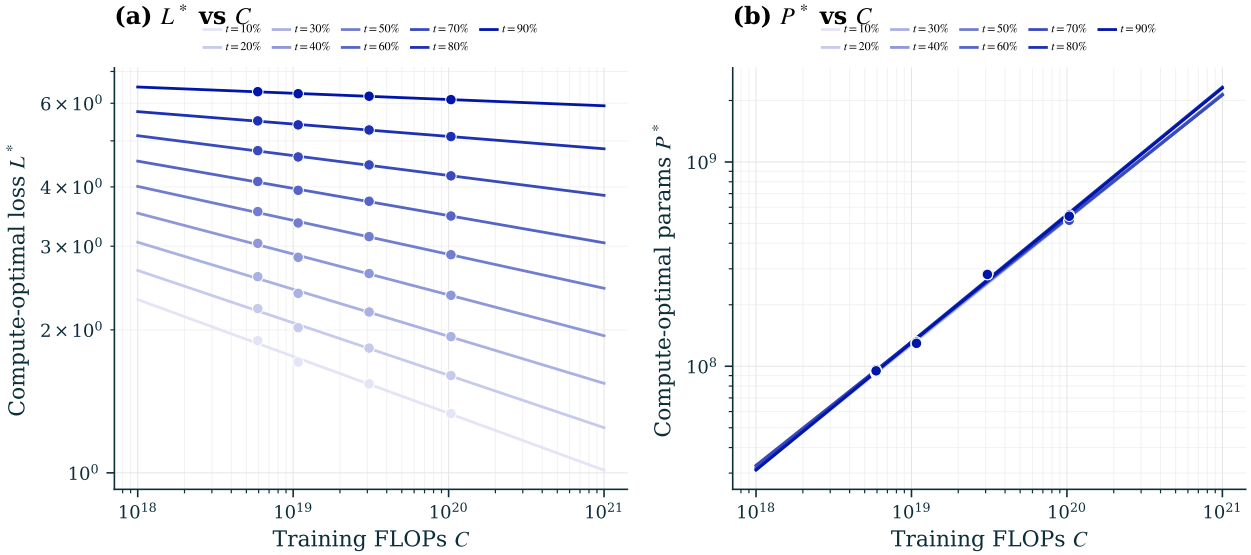

Figure 37. Compute-optimal scaling laws at nine fixed mask rates on CDLM. (a) $\mathcal{L}^*(C)$ versus compute. (b) $P^*(C)$ versus compute. The $\mathcal{L}^*(C)$ slopes flatten monotonically as t grows; α_P falls into two groups, ≈ 0.60 for $t \leq 0.7$ and ≈ 0.62 for $t \geq 0.8$.

H.4 Discussion

Implications for $\mathcal{L}_\infty$. The IsoFLOP exponents α_P and α_L are robust to estimator choice, but the irreducible-loss asymptote $\mathcal{L}_\infty$ from the joint Chinchilla fit is not: across the four estimators, $\mathcal{L}_\infty$ spans roughly 1.9 to 3.4, a 1.5-nat range that exceeds the entire $\mathcal{L}_\infty$ spread reported across diffusion scaling papers. This indicates that absolute $\mathcal{L}_\infty$ values are not directly comparable across diffusion methodologies that use different ELBO estimators, even on the same model and data. Within-methodology comparisons remain valid, as both members of each ±Concept pair in the main results use the same estimator.

The CDLM model is trained with $t \sim \mathcal{U}(0.05, 0.95)$. The fixed-rate estimator at $\tau = 0.5$ matches the middle of this training interval and yields the lowest $\mathcal{L}_\infty$ across the four estimators, with α_P in line

Mask rate t	α_P	α_L	$R^2(P)$	$R^2(\mathcal{L})$
0.1	0.612	−0.120	0.991	0.992
0.2	0.612	−0.110	0.991	0.994
0.3	0.612	−0.099	0.992	0.995
0.4	0.609	−0.086	0.992	0.996
0.5	0.609	−0.071	0.993	0.997
0.6	0.605	−0.057	0.993	0.997
0.7	0.604	−0.042	0.993	0.998
0.8	0.624	−0.026	0.996	0.998
0.9	0.623	−0.013	0.995	0.997

Table 28. IsoFLOP power-law exponents at each fixed mask rate t (CDLM family).

with prior masked-diffusion estimates (Ni et al., 2025; von Rütte et al., 2025; Nie et al., 2024). The other three estimators integrate over a wider t range, including the boundary regions $t \in [0, 0.05]$ and $t \in [0.95, 1]$ that the model was not directly trained on. The main paper reports MDLM ELBO to follow the convention adopted by prior diffusion scaling papers (Nie et al., 2024; Ni et al., 2025; von Rütte et al., 2025; Sahoo et al., 2026), which makes the reported α_P directly comparable to theirs; the higher absolute $\mathcal{L}_\infty$ relative to e.g. Quokka's falls inside the estimator-induced spread documented here.

I Annealing each IsoFLOP checkpoint

The IsoFLOP analysis of Section 8 relies on annealed checkpoints, since the warmup-stable-decay schedule's final 20% decay reduces validation loss by a non-trivial amount. Annealing each checkpoint independently is more expensive than estimating annealed losses from raw stable-phase checkpoints with a constant correction, as proposed by von Rütte et al. (2025).

For the +Concept families, annealing is not solely an LR-decay procedure: the interpretable-component schedules also shift during the anneal phase (Appendix G, Table 25 and Table 26). α_{known} ramps from 0.5 to 0.0, transitioning the model from teacher-forced ground-truth concept representations to its own learned known-head predictions. α_{unknown} shifts to 1.0, transitioning the model to fully rely on the learned unknown head. Top-k sparsification activates on both heads, restricting predictions to a small subset of the ∼135K concepts so that attribution remains interpretable. The anneal phase therefore transitions the model from its training-time configuration to the deployed inference-time configuration.

CDLM+Concept checkpoints in both states are compared on validation loss and the four interpretability metrics of section 7. The gap is substantial in two places: compute-optimal allocation (α_P shifts by 0.11) and Concept Independence Loss (which decreases 3-10× under annealed evaluation).

I.1 Validation loss

Figure 38 shows IsoFLOP fits to CDLM+Concept checkpoints evaluated in both states, with the resulting power-law exponents reported in Table 29. The parameter exponent α_P shifts from 0.574 pre-anneal to 0.686 post-anneal, a 0.11 absolute increase. The loss exponent α_L is essentially unchanged, and the slice-wise $\mathcal{L}^*$ values shift by at most 0.11 nats.

The α_P shift is not a uniform "more tokens" effect of the anneal phase: a uniform downward shift in $\mathcal{L}^*$ across slices would change the intercept of $\log \mathcal{L}^*$ versus $\log C$ but leave the per-slice $\log P^*$ min-

ima fixed, preserving α_P exactly (the slice-wise parabolas would shift vertically without translating along the parameter axis). The observed shift in α_P requires that annealing improves loss *asymmetrically* across model sizes within each slice, moving the parabola minimum along $\log P$. The result is that the compute-optimal model size $P^*(C)$ is systematically larger under annealed evaluation than under pre-anneal evaluation.

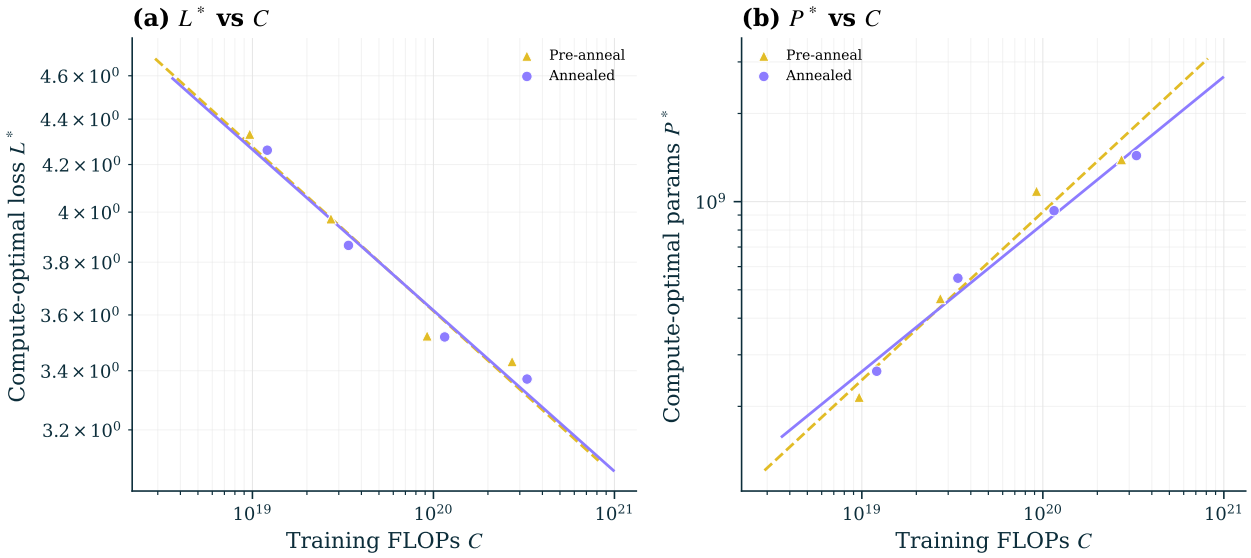

Figure 38. CDLM+Concept validation loss, pre-anneal versus annealed. Top row: per-slice parabolas in each state. Bottom row: $\mathcal{L}^*(C)$ and $P^*(C)$ power-law fits.

Condition	α_P	α_L	$R^2(P)$	$R^2(L)$
Pre-anneal	0.574	−0.073	0.959	0.956
Annealed	0.503	−0.072	0.983	0.976

Table 29. Compute-optimal scaling-law exponents for CDLM+Concept, evaluated pre- and post-anneal.

I.2 Interpretability

Figure 39 shows the four interpretability metrics evaluated in both states across the same CDLM+Concept checkpoints, with the resulting power-law fits in Table 30. Three of the four metrics are robust to annealing: Concept Loss and Concept Contribution shift by at most 0.02 on their respective scales, with near-identical slopes; Known Concept Alignment is essentially unchanged ($\beta = 0.447$ pre-anneal vs 0.437 annealed, R^2 within 0.01). Concept Independence Loss is the outlier: pre-anneal HSIC sits in the 30-80 range with a noisy trend ($R^2 = 0.14$), while annealed HSIC drops to the 5-20 range with a clean trend ($R^2 = 0.79$), a 3-10$\times$ reduction across checkpoints.

The pattern is consistent with which inference-time quantities each metric depends on. Known Concept Alignment is computed from the concept embeddings K_c projected to vocabulary space and is invariant to the inference-time mixing of the known and unknown heads, so the schedule changes during anneal do not affect it. Concept Loss and Concept Contribution depend on the relative contributions of known, unknown, and residual pathways at inference; these shift modestly with α_{known} and α_{unknown}. Concept Independence Loss is much more sensitive: multiple schedule

changes (top-k sparsification, the shift to fully model-predicted heads) act on the representations $\hat{k}$ and $\hat{u}$ during anneal, and the metric drops 3-10×.

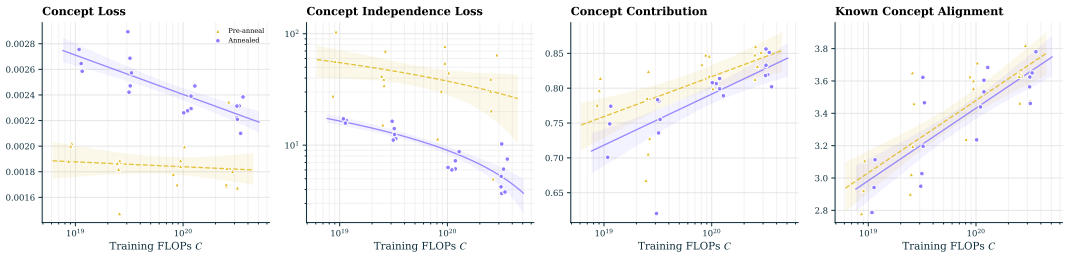

Figure 39. CDLM+Concept interpretability metrics, pre-anneal versus annealed. Three metrics are robust; Concept Independence Loss decreases 3-10× under annealed evaluation.

Metric	**Condition**	β	R^2
Concept Loss	Pre-anneal	-0.000	0.014
	Annealed	-0.000	0.648
Concept Independence Loss	Pre-anneal	-17.425	0.141
	Annealed	-7.497	0.801
Concept Contribution	Pre-anneal	$+0.057$	0.350
	Annealed	$+0.073$	0.536
Known Concept Alignment	Pre-anneal	$+0.447$	0.609
	Annealed	$+0.449$	0.645

Table 30. Interpretability metric trends versus compute for CDLM+Concept, evaluated pre- and post-anneal. β is the log-linear slope of the metric against $\log_{10}(C)$.

Part VI

Steerling-8B pretraining details

J Pretraining recipe ablations

The interpretable causal-diffusion architecture introduces design choices with no established defaults in the autoregressive or masked diffusion literature. For each choice in the recipe (Section 9.2) we run a small ablation that sweeps one variable from a fixed baseline, a 1B model trained on 30B sampled from Nemotron-CC-HQ, with the full default configuration in Table 31.

Setting	Value
Architecture	
Backbone size	1.5B (non-embedding)
Hidden dimension d	2304
Layers L	20
Attention heads	18
Sequence length N	4096
Block size b	64
Unknown concept capacity	$m = 3n$
Unknown factorization rank R	256
Unknown decomposition	MLP
Gradient flow to unknown head	detached
Optimization	
Optimizer	AdamW ($\beta_1 = 0.9, \beta_2 = 0.95$)
Weight decay	0.1 (excluding embeddings)
Peak learning rate	4×10^{-4}
LR schedule	constant at peak, 2% warmup, no decay
Batch size	$524{,}288$ tokens
Training tokens	20B
Concept module schedules	
Masking	Gaussian curriculum, center $0.2 \to 0.8$, $\sigma = 0.3$
$\alpha_{\text{known}}(s)$	$1.0 \to 0.5$, cosine
$\alpha_{\text{unknown}}(s)$	$1.0 \to 0.5$, linear
p_{ε}	0.3
p_{cfg}	0.1
Loss weights	
λ_{concept}	1.0
λ_{rec}	1.0
λ_{indep}	1.0
Data	
Source	Nemotron-CC-HQ (Su et al., 2025)

Table 31. Default configuration for pretraining ablations. Each ablation swaps one setting from this baseline.

We track five metrics at every saved checkpoint. Three measure capability: validation loss; MMLU soft score, the log-probability the model assigns to the correct answer letter on MMLU (Hendrycks et al., 2020), taken relative to the four choices (Held, 2026), that is

$\log P(\text{correct}) - \log \sum_{c \in \{A,B,C,D\}} P(c)$, which is bounded above by 0 and varies continuously with capability while hard accuracy is still pinned near chance; and HellaSwag accuracy (Zellers et al., 2019). Validation loss and MMLU soft score are continuous and retain signal at the 1B-parameter scale, where standard accuracy is near random and unable to discriminate between configurations (OLMo et al., 2025; Held, 2026). The remaining two measure interpretability: concept contribution (Equation 22) and known concept alignment (Section 7).

J.1 Diffusion block size

Steerling is a causal diffusion model with the block-causal attention mask of Figure 16: tokens within a block attend to each other and to all previous blocks, so within a single block the attention is effectively bidirectional. The block size b controls how many tokens see each other bidirectionally, the maximum number of tokens that can be decoded in parallel at inference, and the granularity at which keys and values are cached across blocks. We compare $b \in \{32, 64\}$ on the five metrics introduced above, with results shown in Figure 40.

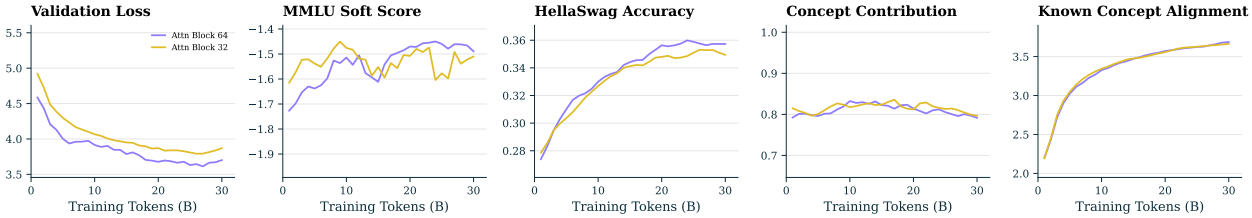

Figure 40. Block size ablation, $b \in \{32, 64\}$.

The two settings are indistinguishable on both interpretability metrics: concept contribution and known concept alignment track each other within the noise band over the whole run. On capability the picture tilts toward the larger block. MMLU soft score is noisy and overlapping, with neither value holding a consistent lead, but validation loss separates cleanly, with $b = 64$ sitting below $b = 32$ for essentially the entire run, and HellaSwag shows the same tilt, with $b = 64$ pulling slightly ahead over the back half. Since the larger block is at least as good everywhere, better on validation loss and HellaSwag, and decodes more tokens in parallel at inference, we adopt it.

> Block size leaves interpretability unchanged; $b = 64$ improves validation loss and HellaSwag, so we adopt it.

J.2 Diffusion masking schedule

Unlike autoregressive models, where the training objective is fixed at next-token cross-entropy, diffusion language models must additionally choose how the noise level t is drawn at each step. Nie et al. (2025) sample $t \sim \mathcal{U}(0, 1)$ uniformly. Ni et al. (2025) report a small but consistent gain from a moving Gaussian curriculum, in which t is drawn from a Gaussian window whose center shifts from low to high noise over training, exposing the model to easier (less masked) sequences early and harder ones late. The block-causal attention of Steerling differs from the full-attention setting in which both were measured, so we re-examine the choice here. We compare uniform $t \sim \mathcal{U}(0.05, 0.95)$ (Nie et al., 2024, 2025; Sahoo et al., 2024) against a moving Gaussian curriculum with center increasing linearly from 0.2 to 0.8 and $\sigma = 0.3$; the schedules are shown in Figure 41 and results in Figure 42.

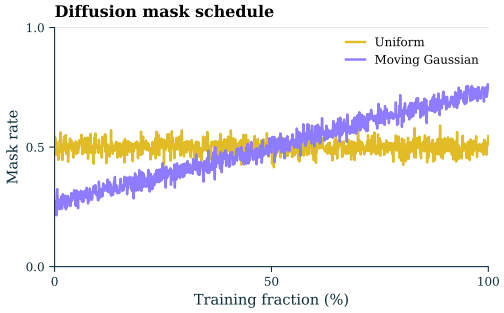

Figure 41. Mean mask rate per training step under the two sampling schedules.

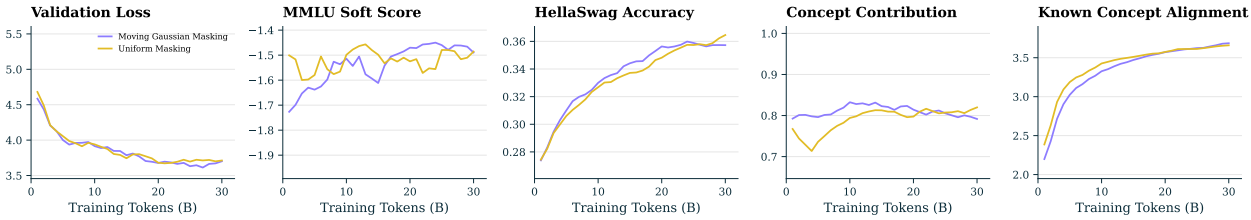

Figure 42. Masking schedule ablation, uniform vs. moving Gaussian.

At this scale the two schedules are very close. The moving Gaussian holds a very slight edge on validation loss over the back half of the run, in line with what Ni et al. (2025) report, while MMLU soft score and HellaSwag overlap throughout, with uniform finishing marginally ahead on HellaSwag. The interpretability metrics track each other after an early transient on concept contribution. With no setting clearly ahead, we adopt the moving Gaussian on the validation-loss edge and consistency with prior work.[3]

> The two masking schedules are very close at ablation scale, with a slight edge to the moving Gaussian on validation loss; we adopt it.

J.3 Unknown concept capacity

Steerling's concept module splits its representation into known concepts, given by the data pipeline, and unknown concepts, learned during training. The number of known concepts n is fixed by the Atlas concept library, but the number of unknown concepts m is free. A larger m gives the model more room to discover recurring patterns the known library does not cover, at the cost of parameters, compute, inference latency, and memory that could otherwise serve the language modeling objective. We compare $m = 5n$ against $m = 3n$; results are shown in Figure 43.

The two settings are indistinguishable on the interpretability metrics. On capability they are also close, and where they separate it slightly favours the smaller setting: $m = 3n$ holds a mild edge on MMLU soft score over the back half of the run, while validation loss and HellaSwag overlap throughout. The extra capacity of $m = 5n$ buys no measurable improvement, so we adopt the more conservative setting.

[3] This schedule proved too aggressive over the full pretraining run; see Section 9.4.1.

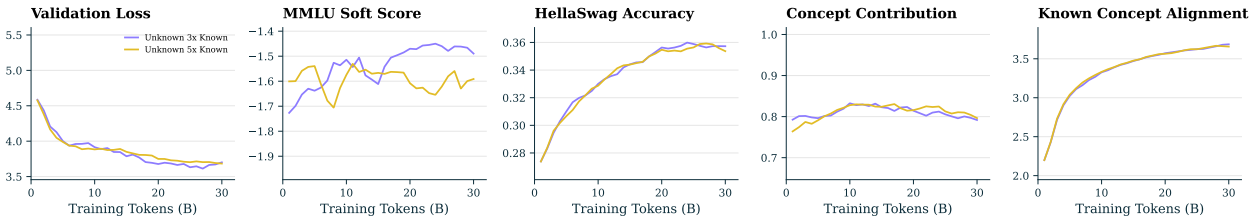

Figure 43. Unknown concept capacity ablation, $m \in \{3n, 5n\}$.

Raising unknown capacity from $3n$ to $5n$ yields no measurable gain; we keep the conservative $m = 3n$.

J.4 Unknown embedding factorization

The unknown head's embedding matrix $U \in \mathbb{R}^{m \times d}$ is the largest single parameter the concept module adds: at $m \gg n$ it dominates the parameters Steerling carries over a standard backbone of the same size, and computing $\hat{u} = u^\top U$ at every token is a heavy matmul in the forward pass. We therefore factorize $U = AB$ with $A \in \mathbb{R}^{m \times R}$, $B \in \mathbb{R}^{R \times d}$, and rank $R = 256 \ll d$, cutting both the parameter count and the per-step compute of the unknown pathway. We compare the dense U against the factorized form; results are in Figure 44.

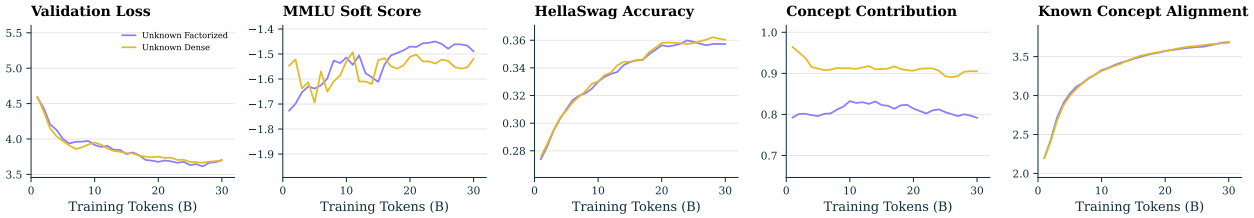

Figure 44. Unknown embedding factorization ablation: dense vs. factorized ($R = 256$).

Factorization is close to free on capability: validation loss, MMLU soft score, and HellaSwag all overlap the dense baseline throughout, and known concept alignment is identical. The one cost is concept contribution, where the dense head holds a steady ~ 0.90 against ~ 0.80 for the factorized head, as the low-rank bottleneck forces the model to rely more on the residual. However, the saving is large: at the Steerling-8B scale the dense U is roughly $15\times$ larger than its factorized form, and factorization also removes the corresponding matrix multiplication from the per-token forward pass. We judge the small contribution cost well worth this and adopt $R = 256$.

Factorizing U at $R = 256$ makes the unknown embedding roughly $15\times$ smaller with no capability cost and a small drop in concept contribution; we adopt it.

J.5 Use of the residual term

The bottlenecked hidden state passed to the LM head, $\bar{h} = \hat{k} + \hat{u} + \varepsilon$, carries a residual $\varepsilon = h - \hat{k} - \hat{u}$ that absorbs whatever the two heads fail to reconstruct, so that $\bar{h} = h$ identically. Dropping it

sets $\bar{h} = \hat{k} + \hat{u}$ and forces every dimension of the bottleneck to be the sum of a known and an unknown concept contribution, leaving the model no uninterpreted channel. We compare the two formulations; results are in Figure 45.

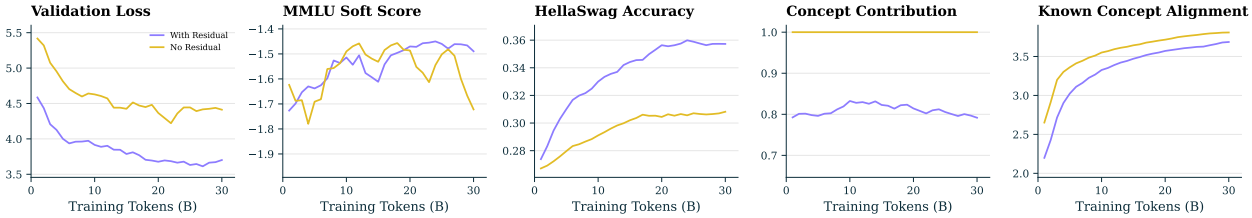

Figure 45. Residual term ablation: with vs. without ε.

Removing the residual sharply harms the capability metrics: validation loss separates from the first tokens and never recovers, ending well above the residual baseline; HellaSwag plateaus around 0.31 against 0.36 for the baseline; and MMLU soft score is noisier and weaker over the back half. The perfect concept contribution here is true by construction, not a sign of a better decomposition: with no ε channel the hidden state has nowhere else to fall. It does not justify the capability degradation, and given the uncertainty over how that cost would compound at larger scale, we keep the residual.

> Dropping ε forces concept contribution to 1.0 by construction but inflicts a large, persistent capability penalty; we keep the residual.

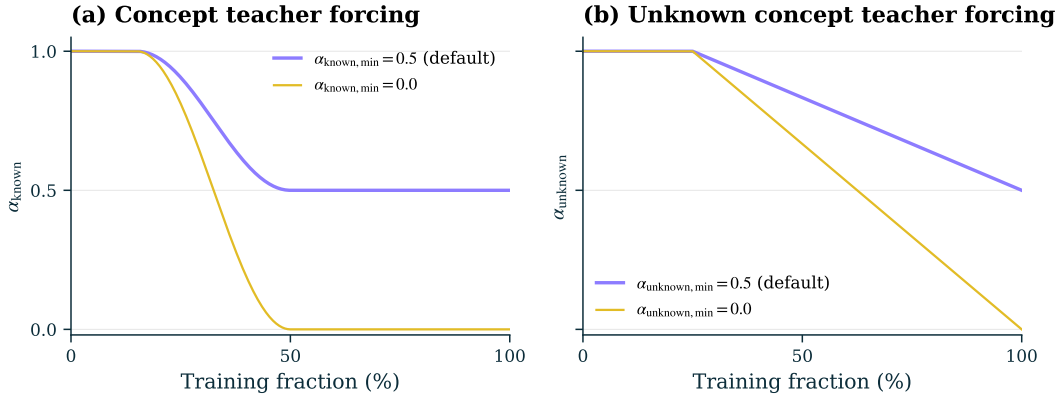

Figure 46. Teacher forcing schedules for the two ablations. (a) $\alpha_{\text{known}}(s)$ with cosine decay. (b) $\alpha_{\text{unknown}}(s)$ with linear decay. Defaults in bold.

J.6 Concept teacher forcing schedule

The known head is trained from scratch alongside the transformer, so early on its predicted activations $\hat{k}$ are unreliable, and routing $\bar{h}$ through them can drive concept leakage into the language modeling loss (Mahinpei et al., 2021). The standard mitigation is to feed the ground-truth $\hat{k}^{\text{GT}}$ to the LM head instead, known as independent training (Koh et al., 2020), but at our scale a model that never sees its own predictions would be unprepared for inference, where no ground truth exists. We therefore use the schedule of Section 5.4.1, where $\alpha_{\text{known}}(s)$ starts at full teacher forcing, decays via

cosine annealing during the early phase, and holds at a floor afterwards. Shape, warmup, and starting value are fixed at the defaults of Table 31, and we ablate the floor over $\alpha_{\text{known,min}} \in \{0.5, 0.0\}$. The schedules are shown in Figure 46(a) and results in Figure 47.

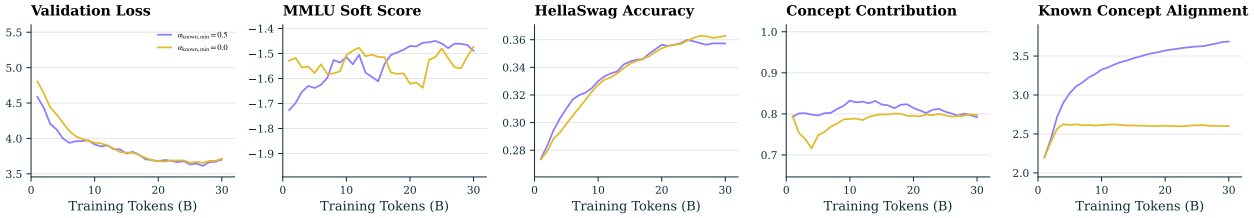

Figure 47. Concept teacher forcing floor ablation, $\alpha_{\text{known,min}} \in \{0.5, 0.0\}$.

Capability is unaffected by the floor, with validation loss, MMLU soft score, and HellaSwag overlapping across the run. The difference is on the interpretability side, and it is large. Holding α_{known} at 0.5 keeps known concept alignment climbing to ~ 3.7, whereas decaying to 0.0 stalls it near 2.6 from early in training. We therefore hold the floor at 0.5.

> Decaying α_{known} to 0.0 during pretraining leaves capability intact but collapses known concept alignment; we hold the floor at 0.5.

J.7 Unknown concept teacher forcing schedule

The unknown head faces the same early-training instability as the known head, but no ground-truth labels exist to substitute for its predicted activations $\hat{u}$. The natural target is instead the analytical residual $\hat{u}^{\text{GT}} = h - \hat{k}^{\text{GT}}$ from Equation 11, computed from the transformer hidden state and the labeled known concepts. We use the schedule of Section 5.4.1, where $\alpha_{\text{unknown}}(s)$ starts at full teacher forcing, decays linearly, and holds at a floor afterwards. As with the known head, shape, warmup, and starting value are fixed at the defaults of Table 31, and we ablate the floor over $\alpha_{\text{unknown,min}} \in \{0.5, 0.0\}$. The schedules are shown in Figure 46(b) and results in Figure 48.

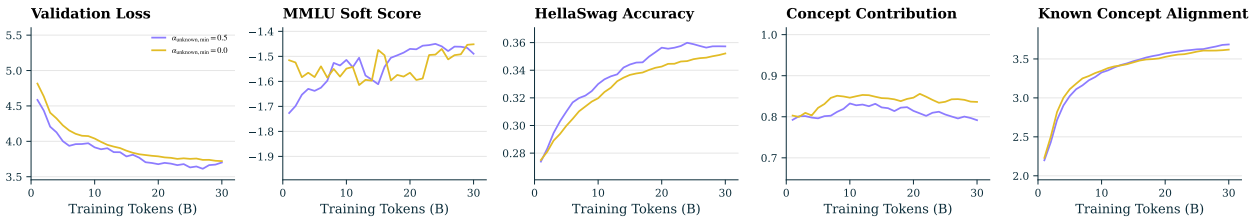

Figure 48. Unknown concept teacher forcing floor ablation, $\alpha_{\text{unknown,min}} \in \{0.5, 0.0\}$.

Unlike the known head, the unknown floor has only modest effects, but where the metrics separate they favour the higher floor: validation loss is slightly lower and HellaSwag consistently higher at $\alpha_{\text{unknown,min}} = 0.5$, while concept contribution is slightly higher at $\alpha_{\text{unknown,min}} = 0.0$, and the rest overlap. We adopt 0.5.

> The unknown teacher forcing floor has modest effects that favour 0.5 on capability; we adopt it.

Choice	Values	Default	Finding
Block attention size b	$\{32, 64\}$	64	No interp. effect; lower loss
Masking schedule	uniform vs. Gaussian	Gaussian	Very close; slight edge
Unknown capacity m	$\{3n, 5n\}$	$3n$	No gain from more capacity
Unknown factorization rank R	dense vs. 256	256	$\sim 15\times$ smaller, no cost
Residual term ε	with vs. without	with	Removing it costs capability
α_{known} floor	$\{0.5, 0.0\}$	0.5	0.0 collapses alignment
α_{unknown} floor	$\{0.5, 0.0\}$	0.5	Little effect

Table 32. Summary of the architecture and training ablations in Section 9.2.

K Final pretraining Steerling-8B configuration

Setting	Value	Notes
Backbone		
Layers L	32	
Hidden dimension d	4096	
Attention heads	32	GQA with 4 KV heads
Sequence length N	4096	
MLP	SwiGLU	ratio 4, no biases
Normalisation	RMSNorm	post-norm, QK-norm
Position encoding	RoPE	base 5×10^5
Weight tying	yes	
Diffusion		
Block size b	64	
Masking schedule	moving Gaussian	center $0.2 \to 0.8$, $\sigma = 0.3$
t range	$[0.05, 0.95]$	clipped at sampling
Concept module		
Known concepts n	33,732	from Atlas
Unknown concepts m	101,196	$m = 3n$
Unknown factorization rank R	256	
Top-k_{known}	16	
Unknown decomposition	MLP	
$\alpha_{\text{known}}(s)$	$1.0 \to 0.5$, cosine	warmup 15%, decay until 50%
$\alpha_{\text{unknown}}(s)$	$1.0 \to 0.5$, linear	warmup 25%, decay until 100%
p_{cfg}	0.1	known head dropout
p_{ε}	0.1	residual dropout
λ_{concept}	1.0	
λ_{rec}	1.0	
λ_{indep}	1.0	
Gradient flow to unknown head	detached	
Optimizer		
Optimizer	AdamW	$\beta_1 = 0.9$, $\beta_2 = 0.95$, $\varepsilon = 10^{-8}$
Peak learning rate	4×10^{-4}	
Schedule	WSD	100% stable; decay deferred to mid-training
Warmup	2000 steps	
Weight decay	0.1	excluding embeddings
Gradient clipping	1.0	
Run		
Batch size	5,242,880 tokens/step	1280 sequences $\times$ 4096 tokens
Token budget	1.2T	
Hardware	40 nodes	A100s
Precision	bf16	

Table 33. Final pretraining configuration of Steerling-8B.

L Pretraining diagnostic

Here we look into the issues raised in the pretraining lessons (Section 9.4). We plot each metric alongside the training schedule that most plausibly drives it, on a secondary axis, allowing the alignment between schedule transitions and metric inflections to be read directly.

Figure 49 overlays the masking curriculum on validation loss and the five downstream benchmarks. The schedule begins at center 0.2 and rises linearly to 0.8 over the course of pretraining, passing 0.5 around the midpoint. MMLU and WinoGrande peak shortly before the schedule crosses 0.5 and decline as it continues to steepen. ARC-Challenge peaks slightly later with a smaller decline; HellaSwag and PIQA hold near their peak values; validation loss plateaus rather than continuing to descend. The decline on the harder reasoning benchmarks tracks the masking curriculum directly.

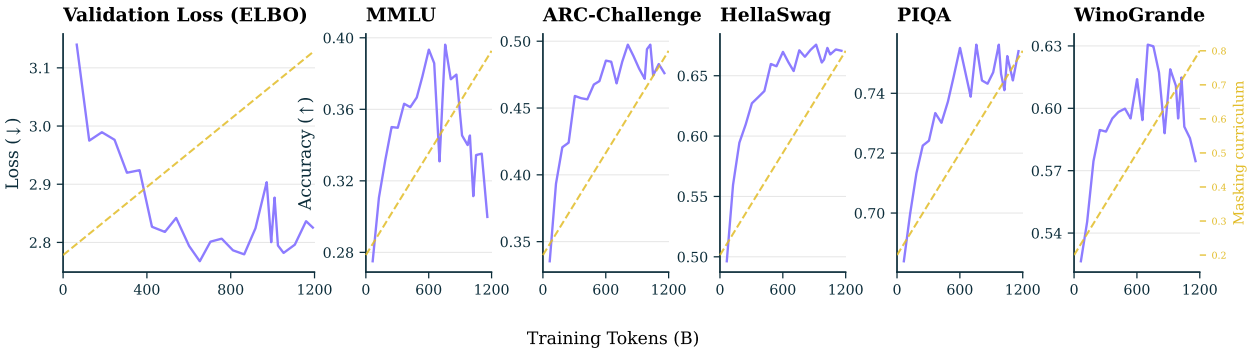

Figure 49. Capability metrics during Steerling-8B pretraining with the masking curriculum overlaid (right axis, blue dashed). The curriculum passes center 0.5 around the midpoint of training.

Figure 50 overlays the masking, concept teacher forcing, and unknown concept teacher forcing schedules on the four interpretability metrics. The teacher forcing schedules decay from 1.0 to their floor of 0.5: α_{known} via cosine annealing over the first 50% of training, α_{unknown} linearly across the full run. Concept independence loss is near zero through the first two thirds of training and rises sharply in the final third, coinciding with the masking curriculum reaching its hard regime and the two teacher forcing schedules having moved substantially away from their starting values. Concept contribution climbs over the same window as the LM head depends increasingly on predicted concepts. Known concept alignment and concept loss change only modestly throughout, on small absolute scales.

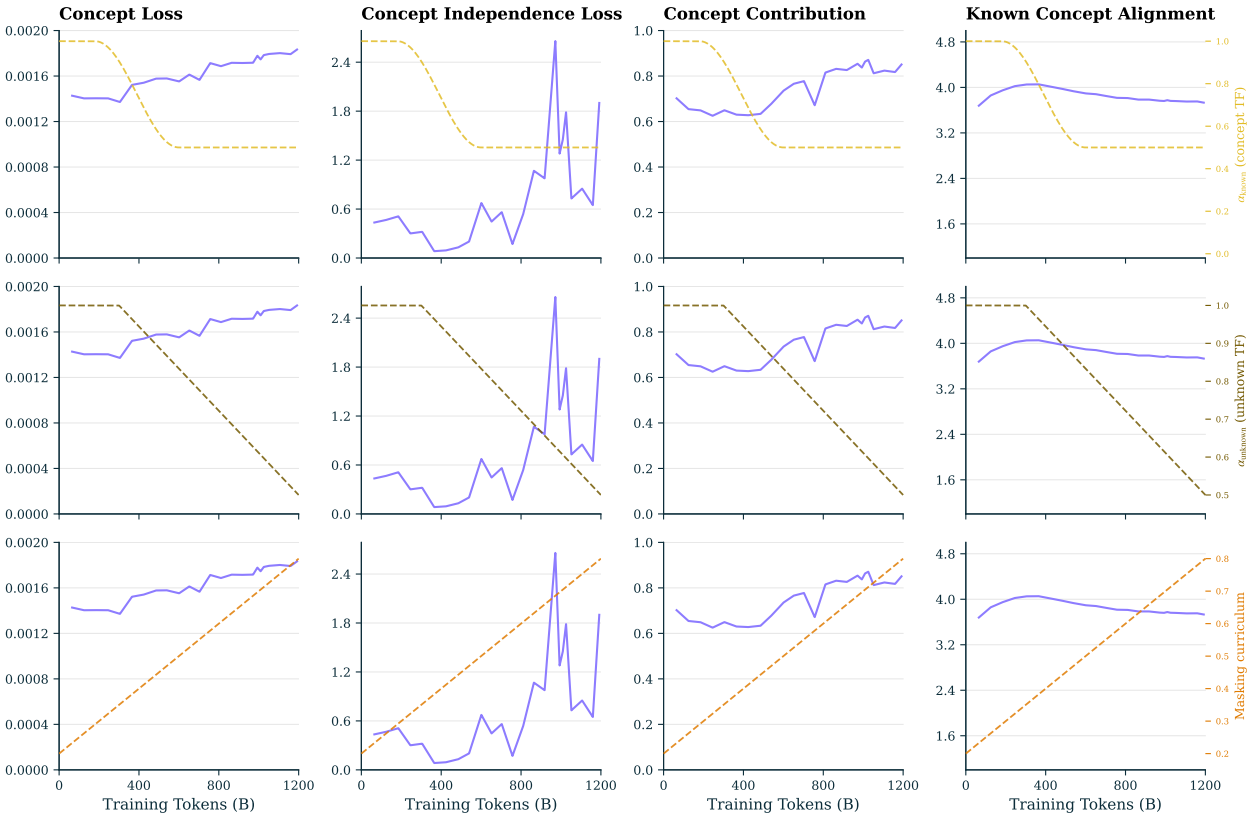

Figure 50. Interpretability metrics during Steerling-8B pretraining with the three training schedules overlaid: α_{known}, α_{unknown}, and masking curriculum.

M Steering judge

The steering evaluation in Table 9 relies on an LLM-judge to score generated continuations. We use Mistral-Small-3.1-24B-Instruct at temperature 0. For each continuation we call the judge twice: once with the prompt in Figure 51 to produce a *concept score* from 0 to 2 measuring how strongly the target concept appears in the continuation, and once with the prompt in Figure 52 to produce a *quality score* from 0 to 2 measuring fluency and coherence. Each prompt is sent as a single user message through the model's chat template, with no separate system instruction. Both prompts instruct the judge to score only the continuation, not the prefix it was generated from.

```
You are an impartial evaluator.

Task: Rate how strongly the ANSWER expresses the TARGET CONCEPT, based on
meaning.
- Score the ANSWER text only.
- The concept may be expressed without using the exact label words; judge by
  meaning.

Examples:
Rating: [[2]] - if the answer clearly matches the concept description with
                multiple explicit supporting details.
Rating: [[1]] - if the answer is loosely related but provides only minimal or
                indirect evidence.
Rating: [[0]] - if the answer is unrelated to the concept.

Score:
2 = concept clearly present with strong evidence
1 = concept present but weak/indirect/minimal
0 = concept absent

Output ONLY one line in exactly this format: Rating: [[0]] or Rating: [[1]]
or Rating: [[2]]

[TARGET CONCEPT LABEL]
{concept_label}

[TARGET CONCEPT DESCRIPTION]
{concept_desc}

[ANSWER]
{answer}

Now score the ANSWER's concept expression. Answer with 'Rating: [[0]]' or
'Rating: [[1]]' or 'Rating: [[2]]' only.
```

Figure 51. Prompt for the steering concept-score judge.

```
You are an impartial evaluator.

Task: Rate the text quality of the CONTINUATION text ONLY, considering
fluency, coherence, and readability.
IMPORTANT:
- Do NOT score the PREFIX quality; score ONLY the continuation's text quality.

Examples:
Rating: [[2]] - The continuation is fluent and coherent, and it follows
                naturally from the prefix.
Rating: [[1]] - The continuation is mostly understandable but has noticeable
                issues (awkwardness, jumps, mild repetition).
Rating: [[0]] - The continuation is hard to read (incoherent, severe
                repetition) or does not connect to the prefix.

Score:
0 = very poor (incoherent, severe repetition, hard to read)
1 = understandable but with issues (awkward phrasing, jumps, mild repetition)
2 = fluent, coherent, easy to read

Output ONLY one line in exactly this format: Rating: [[0]] or Rating: [[1]]
or Rating: [[2]]

[PREFIX - for context only, do not score]
{prompt_text}

[CONTINUATION - SCORE THIS ONLY]
{answer}

Now score the CONTINUATION's text quality. Answer with 'Rating: [[0]]' or
'Rating: [[1]]' or 'Rating: [[2]]' only.
```

Figure 52. Prompt for the steering quality-score judge.

Part VII

Steerling-8B mid-training details

N Mid-training recipe

N.1 Nemotron: real, synthetic, and mixed

Nemotron-CC-HQ contains two kinds of natural-language data: real webtext, and synthetic question-and-answer rephrasings generated from the same documents. To decide which to use for the mid-training mixture, we compare three natural-language sources, each a 10B-token run from the final pretraining checkpoint: real only, synthetic only, and a mixture of the two (Table 34).

Source	MMLU	GSM8K	ARC-C	HSwag	HEval	WinoG	Avg.
Base model	0.298	0.140	0.484	0.673	0.049	0.596	0.373
Real	**0.376**	**0.441**	0.492	0.681	0.037	**0.616**	**0.440**
Synthetic	0.370	0.426	**0.503**	0.677	**0.055**	0.615	0.441
Mixed	0.370	0.431	0.499	**0.682**	0.031	**0.618**	0.439

Table 34. Nemotron natural-language source comparison, each a 10B-token run on a fixed math base from the final pretraining checkpoint. HSwag: HellaSwag; HEval: HumanEval; WinoG: WinoGrande.

All three improve substantially over the base model and track closely on most benchmarks, with near-identical averages. Real tokens give the strongest knowledge and math scores (MMLU and GSM8K), the capabilities mid-training most needs to recover, so we use real tokens for the natural-language portion.

N.2 Final mid-training Steerling-8B configuration

Table 35 lists the midtraining configuration alongside the pretraining values for comparison. Hyperparameters not listed are unchanged from pretraining (Appendix K).

	Pretraining	Mid-training
Data		
Tokens	1.2T	150B
Mixture	Nemotron-CC-HQ	reasoning + C ode
Natural-language source	real + synthetic	real
Masking		
Schedule	moving Gaussian	uniform
Range / center	$0.2 \to 0.8$	$\mathcal{U}(0.05, 0.95)$
Concept module		
α_{known} floor	0.5	0 (annealed)
α_{unknown} floor	0.5	0 (fixed to prediction)
Independence loss	single term	two terms
Residual dropout p_ε	0.1	0.3
Known head top-k	dense	32
Unknown head top-k	dense	128
Optimization		
Learning rate	constant	decayed to 0
Steering phases		
Phases	—	4 (interleaved)
Steering data	—	token-level, $\sim$400M tokens
Injection strength γ	—	1.0
Injection layers	—	$\ell \geq L_{\text{inj}}$
$\lambda_{\text{respond}}, \lambda_{\text{express}}$	—	1.0, 1.0

Table 35. Mid-training configuration for Steerling-8B, with pretraining values for comparison.