LLM Pretraining with Continuous Concepts

Jihoon Tack[1,2], **Jack Lanchantin**[1], **Jane Yu**[1], **Andrew Cohen**[1], **Ilia Kulikov**[1], **Janice Lan**[1], **Shibo Hao**[1,3], **Yuandong Tian**[1], **Jason Weston**[1], **Xian Li**[1]
[1]FAIR at Meta, [2]KAIST, [3]UC San Diego
jihoontack@gmail.com, xianl@meta.com

Abstract

Next token prediction has been the standard training objective used in large language model pretraining. Representations are learned as a result of optimizing for token-level perplexity. We propose Continuous Concept Mixing (CoCoMix), a novel pretraining framework that combines discrete next token prediction with continuous concepts. Specifically, CoCoMix predicts "continuous concepts" learned from a pretrained sparse autoencoder and mixes them into the model's hidden state by interleaving with token hidden representations. Through experiments on multiple benchmarks, including language modeling and downstream reasoning tasks, we show that CoCoMix is more sample efficient and consistently outperforms standard next token prediction, knowledge distillation and inserting pause tokens. We find that combining both concept learning and interleaving in an end-to-end framework is critical to performance gains. Furthermore, CoCoMix enhances interpretability and steerability by allowing direct inspection and modification of the predicted concept, offering a transparent way to guide the model's internal reasoning process.

1 Introduction

Recent progress in large language models (LLMs) has revolutionized natural language processing [6, 13] and thus became a core technology in various real-world applications, such as coding assistants [39], search engines [50], and personal AI assistants [14]. Central to these breakthroughs is the simple paradigm of next token prediction, which leverages massive amounts of unlabeled text to uncover rich linguistic patterns [36, 37]. However, natural language tokens are often superficial (e.g., function words like "the" or "a"), necessitating substantial training for models to acquire high-level reasoning and conceptual understanding while also hindering their ability to tackle long-horizon tasks such as planning [25, 2].

To tackle this issue, recent studies have investigated methods that go beyond token-level signals by leveraging richer information to train models. For instance, some approaches target more expressive prediction objectives, such as predicting multiple tokens at once to better capture semantic relationships [16, 11], while others augment the input with rich signals, e.g., self-generated thought tokens [55], or fixed pause tokens [17] prior to next token prediction. Moreover, emerging evidence suggests that LLMs inherently encode high-level concepts and reasoning processes in their latent representations [12, 51], indicating that replacing discrete language tokens with continuous latent representations has promise in improving reasoning efficiency [19]. While token-level modeling remains important for coherent text generation, the key challenge is to enrich or supplement these natural language tokens so that LLMs can learn more abstract reasoning abilities and long-range dependencies.

This raises a key question: can we augment the next token prediction objective to explicitly model concepts in a latent representation space, thereby bridging semantic abstraction and fine-grained token-level guidance?

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Mech Interp

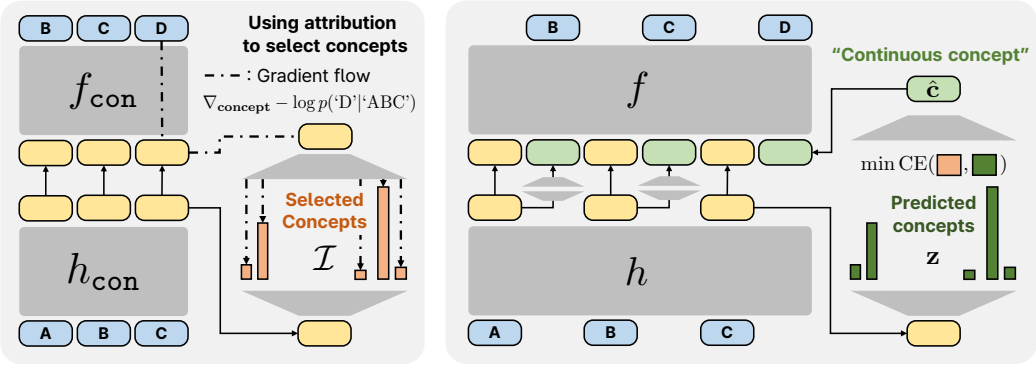


Figure 1: **Overview of CoCoMix.** We use an SAE to extract concepts from a pretrained model's hidden state $h_{\texttt{con}}$ and then select important concepts based on the attribution score (i.e., measuring the influence on the output). These selected concepts are used as labels $\mathcal{I}$ for concept prediction by minimizing the cross-entropy loss $\mathrm{CE}(\cdot, \cdot)$. The predicted concepts $\mathbf{z}$ are then compressed into a compact vector, forming a continuous concept $\mathbf{c}$, which is mixed into the model's hidden state by interleaving with token hidden representations. We demonstrate that CoCoMix is more sample efficient and outperforms standard next-token prediction and knowledge distillation baselines.

To this end, we draw inspiration from recent findings that *Sparse Autoencoders (SAEs)* can effectively isolate meaningful latent features in LLMs by capturing the high-level semantic concepts [10, 5]. As SAEs are trained to reconstruct the model's hidden state with sparsity constraints, it encourages focusing on a compact set of concept dimensions [47]. This makes it possible to highlight the pretrained model's concepts—the core semantic directions that underpin model prediction—while avoiding unnecessary features.

We propose Continuous Concept Mixing (CoCoMix), a novel and effective language modeling framework using continuous concepts. Specifically, we extract semantic concepts using a pretrained SAE and select the most influential ones based on attribution scores, which quantify each concept's influence on the model's output. The model is then trained to predict these selected concepts from its hidden state using a cross-entropy loss. Once multiple concepts are predicted, we compress them into a single *continuous concept* and *mix* (or insert) into the hidden states by interleaving with token embeddings, thereby directly contributing to the next token prediction. An additional benefit is that one can probe or analyze the predicted concepts, enabling controllable generation and improving model interpretability. An overview of CoCoMix is shown in Figure 1.

We demonstrate the efficacy of CoCoMix through extensive evaluations across multiple language modeling benchmarks and pretraining model sizes, ranging from million-scale to billion-scale parameter sizes. For instance, when applied to a 1.38B-sized model, CoCoMix achieves comparable performance with the next token prediction with 21.5% fewer training tokens. Moreover, CoCoMix demonstrates substantial improvements in weak-to-strong supervision scenarios, where concepts extracted from a small model can even be used as ground truth labels to supervise the training of a larger model. Finally, we show that the insertion of compressed concept vectors enables us to steer and control the model by probing the predicted concept during generation.

2 CoCoMix: Continuous Concept Mixing

In this section, we present Continuous Concept Mixing (CoCoMix), a framework that extends next-token prediction with continuous concepts. We first review our problem setup of interest and the concept extraction process using a sparse autoencoder (SAE) in Section 2.1. Then, we describe the concept selection framework and explain how the model predicts and interleaves these concepts into its hidden state to improve language modeling in Section 2.2.

2.1 Prelim: Problem Setup and Sparse Autoencoder

Problem setup. We consider a language-modeling task over a vocabulary $\mathcal{V}$, with a training corpus $\mathcal{D} \subseteq \mathcal{V}^*$, where each sequence $\mathbf{x} = (x_1, x_2, \ldots) \in \mathcal{D}$ is drawn i.i.d. from the data distribution. We require a pretrained language model for concept extraction $\mathcal{M}_{\texttt{con}} = f_{\texttt{con}} \circ h_{\texttt{con}}$ which predicts next token probability $\mathcal{M}_{\texttt{con}}(x_t \mid x_{<t})$. Here, we extract the continuous concepts $\mathbf{c}$ from the hidden state $h_{\texttt{con}}$, where $h_{\texttt{con}}$ outputs the hidden state at a chosen layer of depth $L_{\texttt{con}}$, and $f_{\texttt{con}}$ maps that representation to the next token distribution. We aim to train $\mathcal{M}$ to model the same autoregressive factorization $\mathcal{M}(\mathbf{x}) = \prod_{t=1}^{|\mathbf{x}|} \mathcal{M}(x_t \mid x_{<t}, \mathbf{c}_{<t})$, but by augmenting with continuous concepts $\mathbf{c}$.

Sparse Autoencoder. We first propose to extract high-level concepts from a pretrained model's hidden state. We use an SAE, which maps the input to a high-dimensional sparse activation, with the objective of reconstructing the original input [26]. When applied to an LLM, the SAE decomposes the hidden state into multiple dimensions, each of which can be viewed as a distinct concept capturing semantically meaningful features [53, 5]. We consider the TopK SAE [30], which uses a TopK activation to enforce sparsity.

Formally, for a given input sequence $\mathbf{x}$ and corresponding pretrained model's hidden state at position t, denoted as $\mathbf{h}_t^{\texttt{con}} = h_{\texttt{con}}(\mathbf{x})_t \in \mathbb{R}^{d_{\texttt{con}}}$, SAE consists of a linear encoder $E : \mathbb{R}^{d_{\texttt{con}}} \to \mathbb{R}^C$ and a linear decoder $D : \mathbb{R}^C \to \mathbb{R}^{d_{\texttt{con}}}$, where C is the dimension of the concept space. The reconstruction process of SAE is as:

$$\mathbf{c}_t^{\texttt{pre}} = E\big(\mathbf{h}_t^{\texttt{con}}\big), \;\; \mathbf{c}_t = \mathrm{TopK}\big(\mathbf{c}_t^{\texttt{pre}}\big), \;\; \widehat{\mathbf{h}}_t^{\texttt{con}} = D\big(\mathbf{c}_t\big),$$

where $\mathbf{c}_t^{\texttt{pre}} \in \mathbb{R}^C$ is the pre-activation concept vector, $\mathrm{TopK}(\cdot)$ zeros out all but the largest $K_{\texttt{SAE}}$ entries, and $\widehat{\mathbf{h}}_t^{\texttt{con}}$ is the reconstruction. The SAE is trained by minimizing the reconstruction loss: $\|\mathbf{h}_t^{\texttt{con}} - \widehat{\mathbf{h}}_t^{\texttt{con}}\|_2^2$. By enforcing TopK sparsity, the SAE isolates the most critical dimensions in $\mathbf{c}$ that explain the pretrained model's features. Here, each activated element of $\mathbf{c}_t$ is interpreted as a concept.

2.2 Continuous Concept Mixing

Now, we describe the core training pipeline of CoCoMix, which consists of a concept selection framework (refer to the left figure in Figure 1) and two training steps to learn and utilize continuous concepts (refer to the right figure in Figure 1). First, we select important concepts using the attribution score, which measures the influence of each concept on the output. Then, we propose predicting the selected concepts from the model's hidden state using cross-entropy loss, allowing the model to implicitly learn which concepts should be encoded as a hidden representation. Finally, we utilize the predicted concepts to create a "continuous concept", which is interleaved in the hidden states, enabling the model to explicitly learn to use both the continuous concepts as well as the token hidden states. Intuitively, the model selectively learns which concepts are useful for the next token prediction and how it should *mix* them with the token representations.

Target concept selection using attribution. While the extracted concept $\mathbf{c}_t$ represents the core features of the current input context, it may not directly indicate which concepts matter most for predicting the ground truth next token x_{t+1} [47]. To this end, we propose utilizing *attribution*, a method to measure the causal influence of each concept on the output [3, 45]. Specifically, the attribution score measures the influence based on the local linear approximation of the effect of changing the concept value. In this paper, we use a simple attribution score that measures the influence by multiplying the loss gradient with a given input [44, 43].

Concretely, for a given input $\mathbf{x}$ and corresponding concept $\mathbf{c}_t$, we define the attribution score $\mathbf{a}_t \in \mathbb{R}^C$:

$$\mathbf{a}_t \;=\; \mathbf{c}_t \odot \nabla_{\mathbf{c}_t} - \log f_{\texttt{con}}\big(x_{t+1} | D(\mathbf{c}_t), \mathbf{h}_{<t}\big), \tag{1}$$

where $\odot$ denotes element-wise multiplication. Based on the computed attribution score, we select salient concepts and incorporate them into language modeling.

Predicting the selected concepts. For a given attribution score $\mathbf{a}_t$, we first select the indices of the concept that have a high score and use these as discrete labels for concept prediction. Let $\mathcal{I} = \{i_1, \ldots, i_{K_{\text{attr}}}\}$ be the set of indices corresponding to the top K_{attr} values of $\mathbf{a}_t$, and $\mathbf{h}_t = h(\mathbf{x})_t \in \mathbb{R}^d$ be the model's hidden state of the given input $\mathbf{x}$ at the same token position. Then, we learn to predict these labels using a linear prediction head M that outputs logit $\mathbf{z}_t = M(\mathbf{h}_t) \in \mathbb{R}^C$ by minimizing

the following cross-entropy loss $\mathrm{CE}(\cdot,\cdot)$ is as follows:

$$\mathcal{L}_{\texttt{concept}}(\mathbf{a}_t) \;=\; \frac{1}{K_{\text{attr}}}\sum_{i\in\mathcal{I}} \mathrm{CE}\big(\mathbf{z}_t,\; i\big). \tag{2}$$

Mixing continuous concepts with token embeddings. To encourage the model to internalize the concept more effectively, we propose "mixing" (i.e., interleaving) the predicted concept with the existing token hidden representations. As our model predicts multiple concepts at once, we propose to compress them into a compact vector through a learnable mapping, which we refer to as a continuous concept. This compressed vector is interleaved with token vectors.

Formally, for a given concept prediction logit $\mathbf{z}_t$, we sparsify the logit using TopK activation to predict the concepts, then compress them into a continuous concept vector $\hat{\mathbf{c}}_t \in \mathbb{R}^d$:

$$\hat{\mathbf{c}}_t \;=\; \mathbf{W}\,\mathrm{TopK}(\mathbf{z}_t) + \mathbf{b}, \tag{3}$$

where $\mathbf{W} \in \mathbb{R}^{d\times C}$ and $\mathbf{b} \in \mathbb{R}^d$ project the TopK-sparse vector to a d-dimensional embedding. We then append $\hat{\mathbf{c}}_t$ to the model's hidden sequence $\hat{\mathbf{c}}_t$ as $(\mathbf{h}_1, \hat{\mathbf{c}}_1, \ldots, \mathbf{h}_t, \hat{\mathbf{c}}_t)$, which is fed into the remaining transformer blocks f. Note that this design not only improves the model's performance but also offers interpretability and controllability through analyzing and steering the predicted concept $\mathbf{z}_t$, which can be probed or tuned during the model's generation process. Furthermore, by analyzing the weights of the compression layer $\mathbf{W}$, one can identify which concept is useful for the next token prediction.

This approach shares similarities with intervention techniques in the mechanistic interpretability literature, which modify the hidden state by adding a concept vector to the original hidden state [57, 49]. However, unlike intervention methods that directly manipulate the hidden state, our approach treats the predicted concept as a separate unit of information by interleaving it in the hidden state. This design allows the model to process the concept independently, leveraging the pretrained model's internal reasoning reflected in the prediction.

Training objective. The final training objective for CoCoMix combines the standard next token prediction loss and the concept prediction term with a tunable coefficient λ as follows:

$$\sum_{t=1}^{T-1} -\log f\big(x_{t+1} \mid \mathbf{h}_{\leq t}, \hat{\mathbf{c}}_{\leq t}\big) \;+\; \lambda\mathcal{L}_{\texttt{concept}}(\mathbf{a}_t). \tag{4}$$

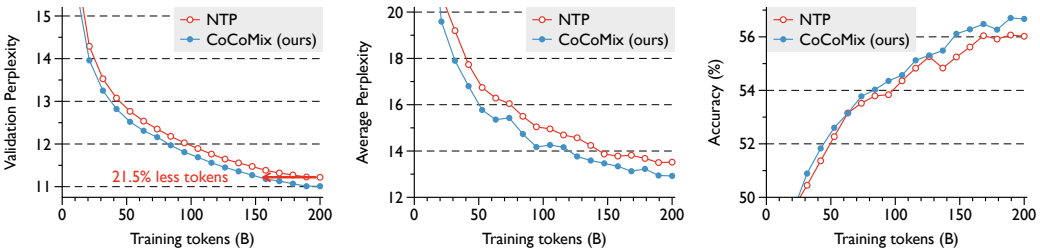
3 Experiments

We provide an empirical evaluation of CoCoMix by investigating the following questions:

- Can CoCoMix improve the performance of next token prediction in LLM pretraining? (Figure 2 and Figure 3)
- Does CoCoMix show improvement in weak-to-strong supervision setup compared to other knowledge distillation methods? (Table 1 and Figure 4)
- Does CoCoMix introduce model interpretability and steerability? (Figure 5)
- How does each proposed component of CoCoMix contribute to the performance? (Figure 6)

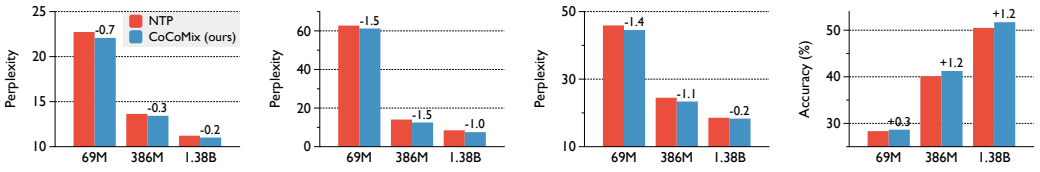
Before answering each question, we outline the experimental protocol (more details in Appendix A).

Training setup. We use a pretrained open-source SAE that is trained on the 124M-sized GPT-2 [15], which has demonstrated strong effectiveness despite a somewhat small model size [8]. Here, we consider training CoCoMix with three different numbers of active parameters, including 69M, 386M, and 1.38B (which is an academic-scale pretraining [1]), with a context length of 1024. For the analysis and ablation study, we mainly conducted experiments on the 69M model. Note that this activated parameter count includes those in the new layer (i.e., the concept predictor), whereas for other baselines, we match the same number of active parameters by increasing the hidden state dimension size d. CoCoMix utilizes fewer FLOPs than Pause token (as discussed in Section 3.3) but more FLOPs than NTP, due to the interleaving of continuous concepts. We use the OpenWebText dataset [37] as the pretraining corpus to use the same distribution used to train $\mathcal{M}_{\texttt{con}}$. All experiments are conducted with 20B training tokens, except for the main experiment in Section 3.1, which uses 200B tokens.

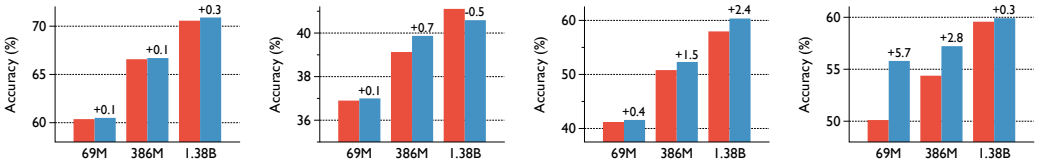


(a) Validation PPL (b) Average downstream task PPL (c) Average downstream task Acc

Figure 2: **CoCoMix vs. NTP performance at different training checkpoints.** Each model contains a total of 1.38B parameters. Each model is trained on the OpenWebText dataset. For CoCoMix, the concepts are extracted from a 124M-sized model (10× smaller than the base model). The plots show improvements in: (a) validation perplexity, (b) average perplexity on LAMBADA, WikiText-103, and (c) average accuracy on HellaSwag, PIQA, SIQA, Arc-Easy, and WinoGrande.



(a) OpenWebText, PPL (↓) (b) LAMBADA, PPL (↓) (c) WikiText-103, PPL (↓) (d) HellaSwag, Acc-n (↑)



(e) PIQA, Acc (↑) (f) SIQA, Acc (↑) (g) Arc-Easy, Acc (↑) (h) WinoGrande, Acc (↑)

Figure 3: **CoCoMix vs. NTP performance at different model sizes.** We consider various model sizes, including 69M, 386M, and 1.38B parameters and train on 200B OpenWebText tokens. We evaluate the models on OpenWebText validation perplexity and downstream datasets LAMBADA, WikiText-103, HellaSwag, PIQA, SIQA, Arc-Easy, and WinoGrande.

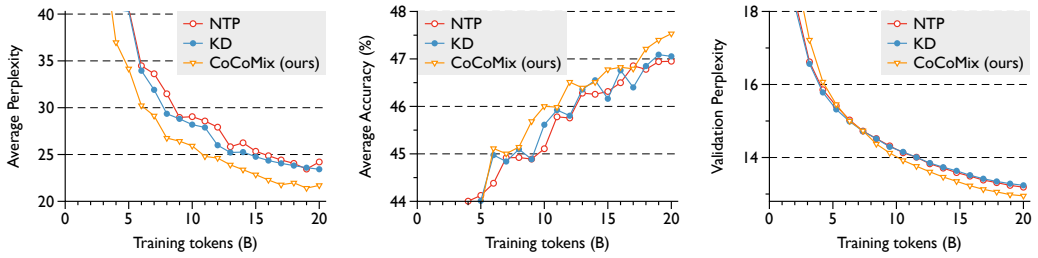
Baselines. We consider the standard pretraining next token prediction (NTP) procedure, and the commonly used distillation method, knowledge distillation (KD) [20], which is widely used in pretraining [18, 41, 46]. We excluded KD baselines that require multiple models (i.e., more than a single teacher model) to be trained [18]. For KD, we minimize the KL divergence between teacher and student output while balancing the KL term with the NTP loss. Note that using synthetic datasets for distillation [21, 52] is excluded due to inefficiency (needs to generate more than 20B tokens).

Evaluation setup. For evaluation, we consider the validation perplexity of the pretraining dataset and 7 downstream tasks to benchmark commonsense reasoning and reading comprehension. This includes LAMBADA [34], WikiText-103 [32], HellaSwag [56], PIQA [4], Social IQA (SIQA) [42], ARC-easy [9], and WinoGrande [40] datasets. We also consider OpenWebMath [35] as a pretraining dataset to demonstrate that concepts learned from a pretrained LLM can still be applied to CoCoMix, even when the model was trained on a different corpus.

3.1 Main Results

In this section, we illustrate two core results: i) the comparison with NTP on a relatively large-scale pretraining setup and ii) the comparison with KD baseline, especially on weak-to-strong supervision scenarios where concepts extracted from a small model are used to guide a larger model.

Improving NTP with CoCoMix at scale. We first present the main result by applying CoCoMix to the NTP. Here, we consider training NTP and CoCoMix on 200B tokens. As shown in Figure 3,



(a) Weak-to-strong: Avg. PPL (b) Weak-to-strong: Avg. Acc (c) Distribution shift: Math domain

Figure 4: **CoCoMix vs. Knowledge Distillation (KD).** For our weak-to-strong supervision setup, we train a 386M model where the teacher of KD (or concept extraction for CoCoMix) is a 124M-sized model: we report (a) average perplexity over OpenWebText, LAMABADA, and WikiText and (b) average accuracy over HellaSwag, PIQA, SIQA, Arc-Easy, and WiniGrande dataset. For (c) a distribution shift setup, we train all methods on OpenWebMath, a math specific pretraining corpus.

Table 1: **CoCoMix vs. Next Token Prediction (NTP) vs. Knowledge Distillation (KD).** We report performance on the OpenWebText (OWT) training set, as well as downstream tasks, including LAMBADA (LAMB), WikiText-103 (Wiki), HellaSwag (HellaS), PIQA, Social Interaction QA (SIQA), Arc-Easy (Arc-E), and WinoGrande (WinoG). We train three different sizes of models where 124M model is used as a teacher. All models are trained on 20B tokens sampled from the OpenWebText dataset. The bold indicates the best result.

Method	Total Params	OWT PPL (↓)	LAMB PPL (↓)	Wiki PPL (↓)	HellaS Acc-n (↑)	PIQA Acc (↑)	SIQA Acc (↑)	Arc-E Acc (↑)	WinoG Acc (↑)	Avg PPL (↓)	Avg Acc (↑)
NTP	69M	25.3	107.6	52.3	27.4	59.4	36.6	39.7	50.7	61.8	42.7
KD	69M	25.2	99.3	51.0	27.4	**59.8**	36.2	**39.8**	50.7	58.5	42.8
CoCoMix	69M	**24.7**	**99.1**	**50.9**	**27.6**	59.5	**37.2**	39.3	**51.0**	**58.2**	**42.9**
NTP	386M	16.3	26.3	29.9	33.6	**64.1**	38.4	47.3	50.9	24.2	46.8
KD	386M	16.4	24.6	**29.1**	33.6	63.6	38.0	**48.5**	51.4	23.4	47.0
CoCoMix	386M	**15.9**	**19.3**	**29.1**	**34.7**	63.6	**39.2**	47.9	**52.3**	**21.4**	**47.5**
NTP	1.38B	14.3	16.6	25.0	38.1	66.1	38.9	50.5	50.0	18.6	48.7
KD	1.38B	14.2	16.6	**24.9**	37.4	66.7	39.0	50.1	52.3	18.5	49.1
CoCoMix	1.38B	**13.9**	**15.4**	**24.9**	**38.4**	**66.9**	**39.5**	**50.8**	**53.0**	**18.1**	**49.7**

CoCoMix consistently and significantly improves the performance of overall downstream tasks on various model sizes. Our results also indicate that larger models (e.g., 386M and 1.38B) can benefit from using concepts extracted from a smaller 124M model, showing effective weak-to-strong supervision. Moreover, as shown in Figure 2, CoCoMix consistently improves the performance over the NTP on a billion-scale model. For instance, CoCoMix achieves similar performance to NTP while using 21.5% fewer tokens, demonstrating high sample efficiency. Finally, it is worth noting that the performance gain of using CoCoMix is increasing over the training steps, demonstrating strong generalization performance.

Comparison with KD baseline. We also compare CoCoMix with KD baseline across multiple scenarios, including (1) a stronger teacher model teaching a smaller student model; (2) weak-to-strong supervision, where a weaker teacher teaches a larger student model; and (3) distribution shift, where the student is trained on a corpus different from the teacher's pretraining distribution. As shown in Table 1, CoCoMix demonstrates improvements over KD in all considered model configurations. In particular, CoCoMix shows significant performance gain in the weak-to-strong supervision setup, e.g., improving average perplexity of 2.8 in 386M, while KD does not show great improvement. This arises because a weaker teacher can introduce noisy or suboptimal knowledge, especially when the student surpasses the teacher in capability [38]. This trend is also observable in Figure 4, where models trained with KD fall behind standard training midway through training as the student outpaces the teacher (especially in the distribution shift scenario). In contrast, CoCoMix selectively utilizes useful concepts, resulting in a consistent performance gain.

3.2 Interpretability and Steerability of CoCoMix

Another core advantage of CoCoMix is its interpretability and model steering. Specifically, as the model is trained to predict concepts in its hidden state, we can analyze which concepts it focuses

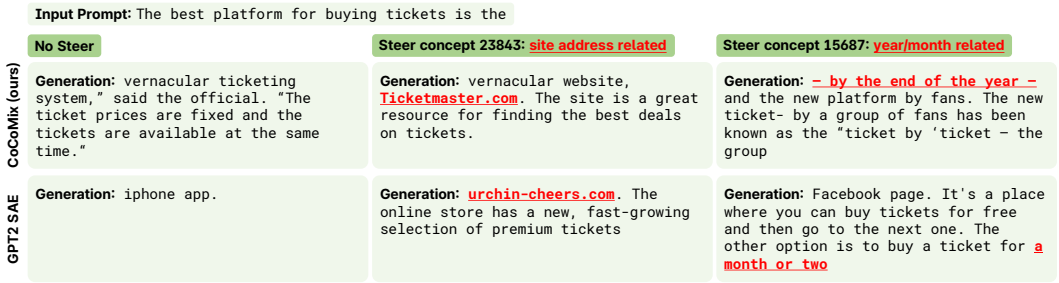


Figure 5: **Qualitative demonstration of the concept steering effect.** CoCoMix and GPT2 models are 350M and 124M parameter transformers, respectively, trained on the OpenWebText dataset. For CoCoMix, we manipulate the predicted concept logit $\mathbf{z}$, while for GPT2, we adjust the SAE concept space $\mathbf{c}$ by increasing the activation of a specific concept index. This illustrates the impact of targeted concept steering on the respective model outputs.

on based on the concept predictions. Furthermore, by amplifying the magnitude of the predicted concept $\mathbf{z}_t$, one can control the output generation of the model. Following Templeton et al. [47], we multiply $\mathbf{z}_t$ of a desired concept element by constants ranging from -10 to 10. To verify whether this steerability works as intended, we steer the activation of the same concept in the pretrained model's SAE latent space $\mathbf{c}$ and confirm whether the output exhibits the corresponding concept. Here, we use a 386M parameter model trained with CoCoMix, where the pretrained model is GPT-2. As shown in Figure 5, when the concept related to "website address" is amplified, both models start generating actual website addresses. This demonstrates that our model has successfully learned the GPT-2 aligned concepts. More examples of steering can be found in Appendix B.3.

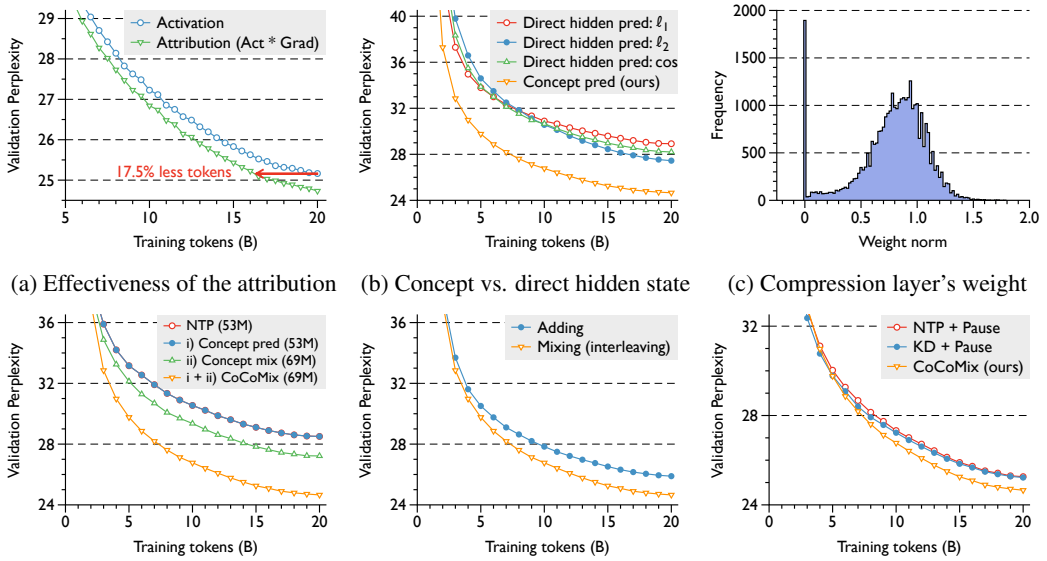
3.3 Analysis of CoCoMix's effectiveness

In this section, we provide a detailed analysis of CoCoMix to validate the effect of each proposed component. Unless otherwise specified, we use a 69M model and train on 20B tokens sampled from the OpenWebText dataset across all methods throughout this section.

Effectiveness of the attribution score. We first analyze whether the attribution score effectively extracts important concepts. To demonstrate this, we train CoCoMix using the activation value $\mathbf{c}_t$ for concept extraction, i.e., $\mathcal{L}_{\texttt{concept}}(\mathbf{c}_t)$ in Equation 2, instead of the attribution score $\mathbf{a}_t$. Remark that the activation value also well reflects the importance of the concept [5]. As shown in Figure 6a, using attribution scores significantly improves performance, improving sample efficiency by 17.5% compared to activation value based selection. We believe it will be an interesting future direction to explore other selection criteria for improving CoCoMix's performance or removing undesirable concepts to reduce bias, e.g., selectively removing unsafe concepts for safe LLM pretraining.

Comparison with direct hidden state predictions. To evaluate the importance of predicting the concept extracted from SAE, we compare CoCoMix with direct hidden state prediction strategies (i.e., predict the full hidden state without projecting into the concept space). To have a comparison under the same architecture as CoCoMix, we replace the concept prediction head M with a two-layer multilayer perceptron (MLP), denoted $g(\cdot)$, which predicts the pretrained LLM's hidden state $\mathbf{h}^{\texttt{con}}$ directly from the hidden state of the model $\mathbf{h}$. The predicted representation, $g(\mathbf{h})$, is then compressed into a continuous embedding for insertion to have the same architecture as CoCoMix. Here, we use continuous loss including, ℓ_1, ℓ_2, and the cosine distance (e.g., $|\mathbf{h}^{\texttt{con}} - g(\mathbf{h})|_2^2$ for ℓ_2) to predict the hidden state. As shown in Figure 6b, direct hidden state prediction leads to a performance drop. We conjecture this to be due to SAE's ability to decompose the latent into semantically meaningful concepts while predicting all hidden states may include noisy components, emphasizing the effectiveness of our method.

Compression layer weight analysis. Now, we analyze the weight of the compression layer $\mathbf{W}$ in Equation 3 to show how CoCoMix utilize the predicted concept. To this end, we visualize the ℓ_2 norm of each concept's weights of 386M CoCoMix: for the weight matrix $\mathbf{W} \in \mathbb{R}^{d \times C}$, where d is the hidden dimension and C is the number of concepts, the norm is defined as: $\|\mathbf{W}_{:,c}\|_2 = \sqrt{\sum_{d=1}^{D} W_{d,c}^2}$. As shown in Figure 6c, we found that a portion of concept weights are near zero, e.g., 5.8% has a norm



(a) Effectiveness of the attribution (b) Concept vs. direct hidden state (c) Compression layer's weight

(d) Component analysis (e) Concept conditioning method (f) Comparison with Pause Token

Figure 6: **Analysis of CoCoMix.** (a) Effectiveness of the attribution score for selecting concepts. (b) Comparison between concept prediction and direct hidden state prediction (i.e., predicting the hidden state with continuous loss rather than discretizing the hidden state with SAE). (c) The sparsity in the compression weight. (d) Component analysis by analyzing the contribution of concept prediction and mixing. (e) Design choices for concept conditioning by comparing adding the concept vector to the original hidden state and mixing (interleaving the concept vector with token hidden representation). (f) Comparison between CoCoMix and the Pause token (i.e., adding learnable tokens).

less than 1e-2. This indicates that CoCoMix ignores these concepts when compressing them into a continuous concept if it is not useful. We conjecture this enabled CoCoMix for strong weak-to-strong supervision as it learned to ignore ineffective concepts extracted from a weak model.

Component Analysis. We analyze the contributions of each component of our method: (a) concept prediction Equation 2, and (b) concept insertion Equation 4. The results in Figure 6d demonstrate that both components are critical for performance improvement. Specifically, applying the concept prediction loss alone yields a modest reduction in perplexity. However, incorporating concept insertion alongside prediction enhances the effectiveness of the loss, resulting in further performance gains. This highlights the role of insertion in enabling the model to leverage the pretrained LLM's latent reasoning effectively. Notably, while concept insertion increases parameter count, it has a limited impact on performance when used alone, emphasizing the critical role of concept prediction.

Continuous concept mixing method. We explored two methods for introducing the continuous concept $\hat{\mathbf{c}}_t$ into the model's hidden state $\mathbf{h}_t$, referred to as concept conditioning. The first method, insertion, insert the continuous concept in front of the token embedding, thus enabling the model to have a mixed input of the token and concept. The other option is intervention, which directly modifies the hidden state by adding the concept vector, i.e., $\mathbf{h}_t \leftarrow \mathbf{h}_t + \hat{\mathbf{c}}_t$. As illustrated in Figure 6e, both methods enhance pretraining performance (compared to NTP in Figure 6d), highlighting the importance of concept conditioning, where the insertion method performed better. By introducing a distinct concept vector, the insertion method enables the model to explicitly recognize and effectively utilize concepts during generation, enhancing its overall performance.

CoCoMix vs. pause tokens. Furthermore, we consider an additional baseline that jointly uses pause token [17]. Specifically, the pause token uses an additional learnable token that is inserted into the token embedding, enabling the LLM to think more before predicting the next token, which is similar to our continuous concept insertion. To this end, we insert the pause token for every input token on the same hidden state layer as CoCoMix to ensure comparable computation. Moreover, we also consider training the pause token with KD. As shown in Figure 6f, CoCoMix consistently outperforms the pause token, indicating our inserted (or interleaved) continuous concept indeed consists of useful information to improve the performance.

Computational cost analysis. One possible concern of CoCoMix is the increased computation due to the expanded token sequence in upper layers. To this end, we report the total FLOPs required for training CoCoMix and baselines, including the NTP and the Pause token. As shown in Table 2, CoCoMix introduces a modest FLOPs increase compared to NTP, but remains more efficient than the Pause token while achieving better perplexity in both low-scale and large-scale settings. Furthermore, it is worth noting that CoCoMix with 140.7B tokens matches the perplexity of NTP trained with 200B tokens, using 17.8% fewer FLOPs, highlighting its improved efficiency–performance trade-off.

Table 2: **Computational cost.** Training FLOPs ($\times 10^{18}$) and validation perplexity (Val. PPL) comparison of NTP, Pause token, and CoCoMix, using a 69M model.

Method	Train Tokens	FLOPs	Val. PPL
NTP	20 B	2.88	25.3
+ Pause token	20 B	3.48	25.1
CoCoMix	20 B	3.37	**24.7**
NTP	200 B	28.83	22.7
CoCoMix	141 B	**23.69**	22.7
CoCoMix	200 B	33.68	**22.0**

4 Related Work

Beyond token-level guidance for language modeling. While next token prediction (NTP) remains the standard paradigm, recent works explore guidance beyond language tokens. For instance, some methods explore a better target, such as leveraging multi-token predictions to capture long context dependencies [16, 11] or predicting sequence embedding [27]. Additionally, methods explore new types of inputs, e.g., using latents [19] or self-generated thought as inputs [55], which have shown improving reasoning capabilities. Only recently, concept-level modeling using local encoder-decoder architectures has also been explored to represent a language at a higher abstraction level [24]. Other methods add extra tokens in the input space to increase computation at inference time [33, 48, 17, 23]. In contrast, we propose a pretraining approach that integrates NTP with continuous concepts, connecting high-level semantics with fine-grained token guidance.

Sparse Autoencoders (SAEs). SAEs extend the autoencoder by enforcing sparsity constraints in the latent space [26]. The features learned by SAEs are often interpretable and disentangled, making them useful across various domains, including language modeling [5]. Additionally, SAEs have gained attention in mechanistic interpretability due to their ability to capture coherent semantic concepts [31]. This property has enabled practical advancements in identifying and manipulating semantic concepts and facilitating steering for controlled model outputs [28]. Among SAE variants, TopK SAEs [30] enforce explicit sparsity using a TopK activation function, demonstrating effectiveness even for large models [15]. In this work, we leverage SAE and, to the best of our knowledge, are the first to apply it to LLM pretraining, achieving strong performance while enhancing the interpretability and controllability of the trained model.

Knowledge distillation (KD). Our method can also be related to KD, i.e., transfers the expertise of a teacher model to a student model to enhance performance [20, 54], as CoCoMix extracts high-level semantic features from a pretrained model. Recently, KD for LLMs has garnered increasing attention, leveraging knowledge from a teacher to improve the generative and encoding capabilities of a student [41, 22]. Especially, applying KD to LLM pretraining remains challenging due to the massive token scales (billions to trillions), forcing most current methods to resort to naive token-level probability matching [46, 18]. Additionally, while pretrained models contain a vast amount of learned information and are thus beneficial to use, reusing knowledge from smaller teacher models remains challenging [7]. In this work, we show CoCoMix can even leverage the concept extracted from small models to train a large model showing weak-to-strong supervision.

5 Conclusion

We propose CoCoMix, a new LLM pretraining framework that augments next token prediction with continuous concepts. By leveraging concepts extracted from a pretrained SAE as targets, our model predicts both the next token and the associated concept. Predicted concepts are then compressed into a continuous concept vector, which is then mixed into the hidden state. This approach enhances interpretability and controllability by enabling direct probing of the distilled concepts. Experimental results show that CoCoMix consistently improves performance across benchmarks, particularly in challenging generalization scenarios such as weak-to-strong supervision.

References

[1] Z. Allen-Zhu. Physics of language models: Part 4.1, architecture design and the magic of canon layers, 2025.

[2] G. Bachmann and V. Nagarajan. The pitfalls of next-token prediction. In *International Conference on Machine Learning*, 2024.

[3] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K.-R. Müller. How to explain individual classification decisions. *The Journal of Machine Learning Research*, 2010.

[4] Y. Bisk, R. Zellers, J. Gao, Y. Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *AAAI Conference on Artificial Intelligence*, 2020.

[5] T. Bricken, A. Templeton, J. Batson, B. Chen, A. Jermyn, T. Conerly, N. Turner, C. Anil, C. Denison, A. Askell, R. Lasenby, Y. Wu, S. Kravec, N. Schiefer, T. Maxwell, N. Joseph, Z. Hatfield-Dodds, A. Tamkin, K. Nguyen, B. McLean, J. E. Burke, T. Hume, S. Carter, T. Henighan, and C. Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

[7] C. Burns, P. Izmailov, J. H. Kirchner, B. Baker, L. Gao, L. Aschenbrenner, Y. Chen, A. Ecoffet, M. Joglekar, J. Leike, et al. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision. *arXiv preprint arXiv:2312.09390*, 2023.

[8] M. Chaudhary and A. Geiger. Evaluating open-source sparse autoencoders on disentangling factual knowledge in gpt-2 small. *arXiv preprint arXiv:2409.04478*, 2024.

[9] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[10] H. Cunningham, A. Ewart, L. Riggs, R. Huben, and L. Sharkey. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023.

[11] DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[12] Y. Deng, K. Prasad, R. Fernandez, P. Smolensky, V. Chaudhary, and S. Shieber. Implicit chain of thought reasoning via knowledge distillation. *arXiv preprint arXiv:2311.01460*, 2023.

[13] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[14] D. Gao, L. Ji, L. Zhou, K. Q. Lin, J. Chen, Z. Fan, and M. Z. Shou. Assistgpt: A general multi-modal assistant that can plan, execute, inspect, and learn. *arXiv preprint arXiv:2306.08640*, 2023.

[15] L. Gao, T. D. la Tour, H. Tillman, G. Goh, R. Troll, A. Radford, I. Sutskever, J. Leike, and J. Wu. Scaling and evaluating sparse autoencoders. *arXiv preprint arXiv:2406.04093*, 2024.

[16] F. Gloeckle, B. Y. Idrissi, B. Rozière, D. Lopez-Paz, and G. Synnaeve. Better & faster large language models via multi-token prediction. In *International Conference on Machine Learning*, 2024.

[17] S. Goyal, Z. Ji, A. S. Rawat, A. K. Menon, S. Kumar, and V. Nagarajan. Think before you speak: Training language models with pause tokens. In *International Conference on Learning Representations*, 2024.

[18] Y. Gu, H. Zhou, F. Meng, J. Zhou, and M. Huang. Miniplm: Knowledge distillation for pre-training language models. *arXiv preprint arXiv:2410.17215*, 2024.

[19] S. Hao, S. Sukhbaatar, D. Su, X. Li, Z. Hu, J. Weston, and Y. Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

[20] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[21] Y. Kim and A. M. Rush. Sequence-level knowledge distillation. In *Annual Conference of the Association for Computational Linguistics*, 2016.

[22] J. Ko, S. Kim, T. Chen, and S.-Y. Yun. Distillm: Towards streamlined distillation for large language models. In *International Conference on Machine Learning*, 2024.

[23] J. Lanchantin, S. Toshniwal, J. Weston, S. Sukhbaatar, et al. Learning to reason and memorize with self-notes. *Advances in Neural Information Processing Systems*, 36, 2024.

[24] LCM, L. Barrault, P.-A. Duquenne, M. Elbayad, A. Kozhevnikov, B. Alastruey, P. Andrews, M. Coria, G. Couairon, M. R. Costa-jussà, et al. Large concept models: Language modeling in a sentence representation space. *arXiv preprint arXiv:2412.08821*, 2024.

[25] Y. LeCun. A path towards autonomous machine intelligence. *Open Review*, 2022.

[26] H. Lee, A. Battle, R. Raina, and A. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems*, 2006.

[27] H. Lee, D. Kim, J. Jun, S. Joo, J. Jang, K.-W. On, and M. Seo. Semiparametric token-sequence co-supervision. In *Annual Conference of the Association for Computational Linguistics*, 2024.

[28] T. Lieberum, S. Rajamanoharan, A. Conmy, L. Smith, N. Sonnerat, V. Varma, J. Kramár, A. Dragan, R. Shah, and N. Nanda. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2. *arXiv preprint arXiv:2408.05147*, 2024.

[29] I. Loshchilov. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[30] A. Makhzani and B. Frey. K-sparse autoencoders. In *International Conference on Learning Representations*, 2014.

[31] S. Marks, C. Rager, E. J. Michaud, Y. Belinkov, D. Bau, and A. Mueller. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models. *arXiv preprint arXiv:2403.19647*, 2024.

[32] S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.

[33] M. Nye, A. J. Andreassen, G. Gur-Ari, H. Michalewski, J. Austin, D. Bieber, D. Dohan, A. Lewkowycz, M. Bosma, D. Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

[34] D. Paperno, G. Kruszewski, A. Lazaridou, Q. N. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Annual Conference of the Association for Computational Linguistics*, 2016.

[35] K. Paster, M. D. Santos, Z. Azerbayev, and J. Ba. Openwebmath: An open dataset of high-quality mathematical web text. *arXiv preprint arXiv:2310.06786*, 2023.

[36] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. 2018.

[37] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

[38] A. S. Rawat, V. Sadhanala, A. Rostamizadeh, A. Chakrabarti, W. Jitkrittum, V. Feinberg, S. Kim, H. Harutyunyan, N. Saunshi, Z. Nado, et al. A little help goes a long way: Efficient llm training by leveraging small lms. *arXiv preprint arXiv:2410.18779*, 2024.

[39] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[40] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi. Winogrande: An adversarial winograd schema challenge at scale. In *AAAI Conference on Artificial Intelligence*, 2020.

[41] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. In *Advances in Neural Information Processing Systems*, 2019.

[42] M. Sap, H. Rashkin, D. Chen, R. Le Bras, and Y. Choi. Social IQa: Commonsense reasoning about social interactions. In *Conference on Empirical Methods in Natural Language Processing*, 2019.

[43] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. Not just a black box: Learning important features through propagating activation differences. In *International Conference on Machine Learning*, 2016.

[44] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[45] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic attribution for deep networks. In *International Conference on Machine Learning*, 2017.

[46] G. Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

[47] A. Templeton, T. Conerly, J. Marcus, J. Lindsey, T. Bricken, B. Chen, A. Pearce, C. Citro, E. Ameisen, A. Jones, H. Cunningham, N. L. Turner, C. McDougall, M. MacDiarmid, C. D. Freeman, T. R. Sumers, E. Rees, J. Batson, A. Jermyn, S. Carter, C. Olah, and T. Henighan. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.

[48] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[49] Z. Wu, A. Arora, Z. Wang, A. Geiger, D. Jurafsky, C. D. Manning, and C. Potts. Reft: Representation finetuning for language models. In *Advances in Neural Information Processing Systems*, 2024.

[50] D. Xuan-Quy, L. Ngoc-Bich, P. Xuan-Dung, N. Bac-Bien, and V. The-Duy. Evaluation of chatgpt and microsoft bing ai chat performances on physics exams of vietnamese national high school graduation examination. *arXiv preprint arXiv:2306.04538*, 2023.

[51] S. Yang, E. Gribovskaya, N. Kassner, M. Geva, and S. Riedel. Do large language models latently perform multi-hop reasoning? In *Annual Conference of the Association for Computational Linguistics*, 2024.

[52] P. Yu, J. Xu, J. Weston, and I. Kulikov. Distilling system 2 into system 1. *arXiv preprint arXiv:2407.06023*, 2024.

[53] Z. Yun, Y. Chen, B. A. Olshausen, and Y. LeCun. Transformer visualization via dictionary learning: contextualized embedding as a linear superposition of transformer factors. *arXiv preprint arXiv:2103.15949*, 2021.

[54] S. Zagoruyko and N. Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *International Conference on Learning Representations*, 2017.

[55] E. Zelikman, G. Harik, Y. Shao, V. Jayasiri, N. Haber, and N. D. Goodman. Quiet-star: Language models can teach themselves to think before speaking. In *Conference on Language Modeling*, 2024.

[56] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. Hellaswag: Can a machine really finish your sentence? In *Annual Conference of the Association for Computational Linguistics*, 2019.

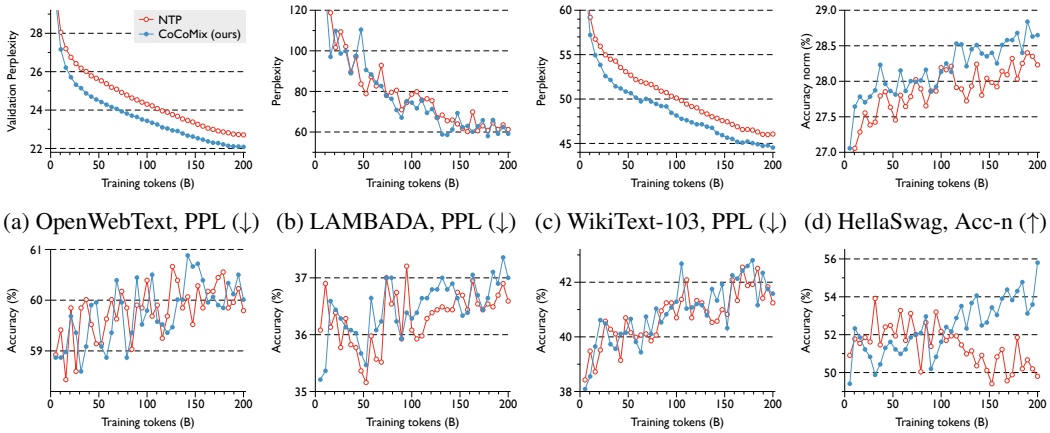
[57] A. Zou, L. Phan, S. Chen, J. Campbell, P. Guo, R. Ren, A. Pan, X. Yin, M. Mazeika, A.-K. Dombrowski, et al. Representation engineering: A top-down approach to ai transparency. *arXiv preprint arXiv:2310.01405*, 2023.

A Experimental Details

Architecture details. Our method and baseline both utilize the GPT-2-based Transformer architecture and tokenizer [37], with a context length of 1024. We consider three model sizes, defined by the number of activated parameters: 69M, 386M, and 1.38B. For CoCoMix, the hidden state dimensions d are 512, 1024, and 2028, with 8, 24, and 24 layers, respectively. The number of attention heads is set to 8, 16, and 16 for the three configurations. Baseline models are configured to match the number of activated parameters by employing the same number of layers and attention heads, with hidden state dimensions of 624, 1072, and 2096, respectively. We leverage an open-source pretrained TopK Sparse Autoencoder (SAE) [15] for concept extraction, where $K_{\texttt{concept}}$ is set to 32 and the concept activation size is 32,768. Consequently, our models introduce an additional $(C + K_{\texttt{concept}}) \times d$ activated parameters on top of the base Transformer parameters. The GPT-2 model (a teacher model for KD and a concept extraction model for CoCoMix) has 12 layers, a hidden dimension size of 768, and 124M parameters. The concept extraction layer $L_{\texttt{con}}$ is configured as the 6th middle layer of GPT-2 for all model sizes. For CoCoMix, the concept prediction is done at the 4th layer for the 69M model and the 6th layer for the larger 386M and 1.38B models. The reported training FLOPs in Table 2 are based on the base model, and show low FLOPs thanks to the high sparsity of the compression layer (only activated with a concept number of 32), which introduces a small cost.

Training details. We mainly followed the training details outlined in [6]. For the main results presented in Section 2, models were trained on 200B tokens over approximately 382K training steps, while other experiments were conducted on 20B tokens. We used OpenWebText as the training dataset to match the same training corpus as GPT-2. The learning rate schedule included a warm-up phase over the first 1/300 of the total training steps, followed by a cosine decay to 10% of the maximum learning rate at the end of training. The maximum learning rates were set to 6e-4, 3e-4, and 2e-4 for the 69M, 386M, and 1.38B models, respectively. The batch sizes were configured to 0.5M, 0.5M, and 1M tokens for the three model sizes. A weight decay of 0.1 was applied across all configurations, and we utilized the AdamW optimizer [29] with $\beta_1 = 0.9$ and $\beta_2 = 0.95$. For training CoCoMix, the concept prediction loss Equation 4 was scaled by a hyperparameter $\lambda = 0.1$, and $K_{\texttt{attri}}$ was set to 4. Here, as the next token prediction (NTP) loss is our primary objective, we chose λ that is smaller than 1 to ensure that the concept prediction loss—which has a similar scale—would not dominate but instead serve as an auxiliary objective. We found that this value empirically worked well with our small-scale model (69M), and thus we consistently used 0.1 throughout all experiments. While further tuning could potentially yield better results, this initial choice was sufficient to clearly demonstrate our method's effectiveness. For training CoCoMix under a 69M transformer for 20B tokens, we have used 16 A100 GPUs for less than 8 hours.

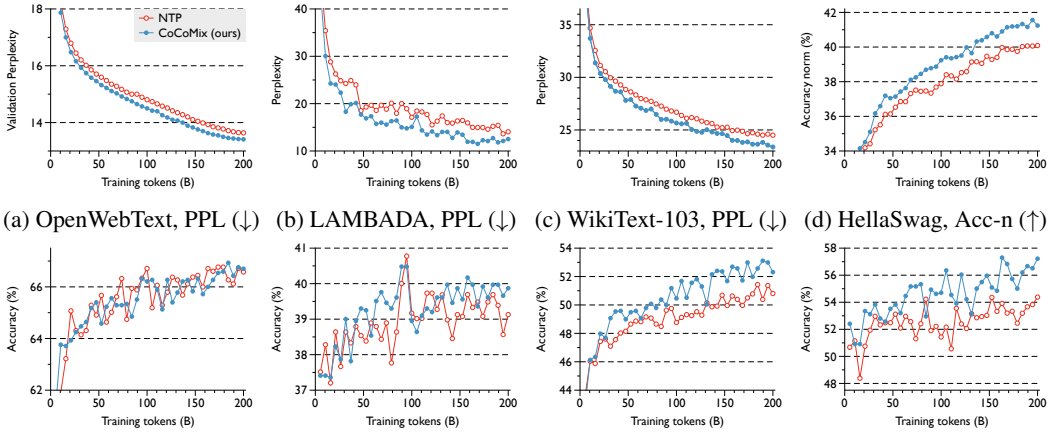
For the KD baseline, we employed the vanilla KD loss, where the output probabilities of the teacher model $\mathcal{M}_{\texttt{con}}$ and the student model $\mathcal{M}$ were matched using the Kullback-Leibler (KL) divergence. Specifically, given an input $\mathbf{x}$, the KD loss is defined as $-\log \mathcal{M}(x_{t+1}|\mathbf{x}) + \lambda_{\text{KD}} \cdot \text{KL}(\mathcal{M}_{\texttt{con}}(\cdot|\mathbf{x}) \| \mathcal{M}(\cdot|\mathbf{x}))$, where $\lambda_{\text{KD}} = 0.1$ consistently demonstrated strong performance.



(a) OpenWebText, PPL (↓) (b) LAMBADA, PPL (↓) (c) WikiText-103, PPL (↓) (d) HellaSwag, Acc-n (↑)

(e) PIQA, Acc (↑) (f) SIQA, Acc (↑) (g) Arc-Easy, Acc (↑) (h) WinoGrande, Acc (↑)

Figure 7: **CoCoMix vs. NTP performance at different training checkpoints on 69M parameter model.** Each model is trained on the 200B tokens sampled from the OpenWebText dataset. The plot shows the result of (a) OpenWebText, (b) LAMBADA, (c) WikiText-103, (d) HellaSwag, (e) PIQA, (f) SIQA, (g) Arc-Easy, and (h) WinoGrande datasets. We use the concepts extracted from a 124M-sized model for training CoCoMix.



(a) OpenWebText, PPL (↓) (b) LAMBADA, PPL (↓) (c) WikiText-103, PPL (↓) (d) HellaSwag, Acc-n (↑)

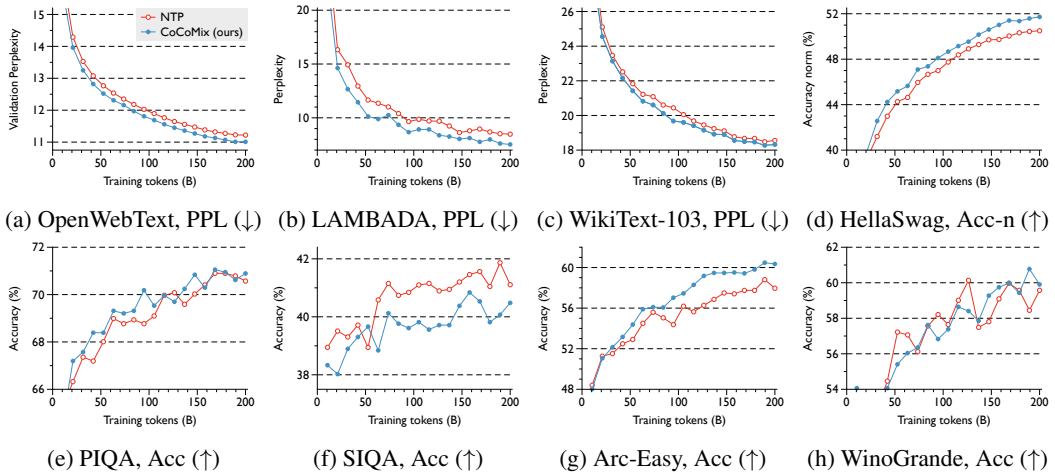
(e) PIQA, Acc (↑) (f) SIQA, Acc (↑) (g) Arc-Easy, Acc (↑) (h) WinoGrande, Acc (↑)

Figure 8: **CoCoMix vs. NTP performance at different training checkpoints on 368M parameter model.** Each model is trained on the 200B tokens sampled from the OpenWebText dataset. The plot shows the result of (a) OpenWebText, (b) LAMBADA, (c) WikiText-103, (d) HellaSwag, (e) PIQA, (f) SIQA, (g) Arc-Easy, and (h) WinoGrande datasets. We use the concepts extracted from a 124M-sized model for training CoCoMix.

B Additional Results

B.1 Detailed Performance During Training

In this section, we present the performance tracking during training on 200B tokens, including validation perplexity and the perplexity and accuracy of various downstream tasks, including LAMBADA, WikiText-103, HellaSwag, PIQA, SIQA, Arc-Easy, and WinoGrande datasets. As shown in Figure 7, Figure 8, and Figure 9, we compare CoCoMix with the next token prediction (NTP) across different active parameter sizes: 69M, 386M, and 1.38B. In most of the graphs, CoCoMix consistently demonstrates performance gains. Notably, our results show that CoCoMix achieves stable improvements in perplexity across all tasks. Furthermore, CoCoMix exhibits sample efficiency;



(a) OpenWebText, PPL (↓) (b) LAMBADA, PPL (↓) (c) WikiText-103, PPL (↓) (d) HellaSwag, Acc-n (↑)

(e) PIQA, Acc (↑) (f) SIQA, Acc (↑) (g) Arc-Easy, Acc (↑) (h) WinoGrande, Acc (↑)

Figure 9: **CoCoMix vs. NTP performance at different training checkpoints on 1.38B parameter model.** Each model is trained on the 200B tokens sampled from the OpenWebText dataset. The plot shows the result of (a) OpenWebText, (b) LAMBADA, (c) WikiText-103, (d) HellaSwag, (e) PIQA, (f) SIQA, (g) Arc-Easy, and (h) WinoGrande datasets. We use the concepts extracted from a 124M-sized model for training CoCoMix.

Table 3: **Comparison of CoCoMix with ablation baselines.** We report validation perplexity on OpenWebText using a 69M-parameter Transformer model.

Method	Validation PPL
NTP (token-only)	25.3
(i) Interleave same token twice	25.1
(ii) Insert concept every $T = 4$ steps	25.0
(iii) Concept-only (no token hidden states)	25.1
CoCoMix (Ours)	**24.7**

for instance, in the 1.38B model, CoCoMix reaches the same OpenWebText validation perplexity as NTP while requiring approximately 43B fewer tokens (a 21.5% improvement in token efficiency).

B.2 More Comparison with Additional Baselines

To further demonstrate the effectiveness of CoCoMix, we consider three additional baselines. Specifically, we compare with: (i) duplicating each token hidden state, resulting in an interleaved sequence of $(\mathbf{h}_1, \mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_2, \cdots, \mathbf{h}_t, \mathbf{h}_t)$; (ii) interleaving continuous concepts every T steps instead of at every time step, yielding $(\mathbf{h}_1, \cdots, \mathbf{h}_T, \hat{\mathbf{c}}_T, \mathbf{h}_{T+1}, \cdots, \mathbf{h}_{2T}, \hat{\mathbf{c}}_{2T}, \cdots)$; where we use $T = 4$ and (iii) using only continuous concepts without token hidden states, i.e., $(\hat{\mathbf{c}}_1, \hat{\mathbf{c}}_2, \cdots, \hat{\mathbf{c}}_t)$.

This comparison isolates key components of CoCoMix and highlights their contributions to overall performance. We present the result in Table 3. First, duplicating token hidden states to match the sequence length of CoCoMix yields worse performance despite higher computational cost (it is worth noting that the copied baseline increases the forward GFLOPs even higher than CoCoMix, 164.46 $\rightarrow$ 169.52 for computing 1024 tokens), indicating that gains are not solely due to sequence length. Second, reducing the interleaving frequency of concept vectors leads to performance degradation, suggesting that frequent integration of semantic signals is crucial. Lastly, a model using only continuous concepts outperforms token-only models but still underperforms CoCoMix, confirming that the synergy between token-level and concept-level representations is essential for optimal effectiveness.

B.3 Additional Steerability Results

Input Prompt: The best platform for buying tickets is the

	No Steer	Steer concept 12783: $ dollar related	Steer concept 19193: Phone related
CoCoMix (ours)	**Generation:** vernacular ticketing system," said the official. "The ticket prices are fixed and the tickets are available at the same time."	**Generation:** vernacular website, which is a great resource for finding out about the event. The event is free to attend, but you will need to pay for your ticket. The cost is $25 for	**Generation:** iphone.The iPhone is a great way to buy tickets to the game. It is easy to use, and it is very fast. You can buy a ticket to a game, pay for
GPT2 SAE	**Generation:** iphone app.	**Generation:** $10.00 price. The price is $10 for a ticket. I bought a $5.50 ticket for $1. The price of a good ticket is a lot more than a	**Generation:** iphone. The iPhone is a great way to get around the iPhone. It's a good way for people to buy a smartphone. The iPhone has a lot of features, but it's not

(a) Main figure prompt, '$ dollar' and 'Phone' concept

Input Prompt: Latent concept modeling is a

	No Steer	Steer concept 15687: year/month related	Steer concept 19193: Phone related
CoCoMix (ours)	**Generation:** technique that allows you to model a complex system in a way that is easy to understand and easy for the user to manipulate.	**Generation:** way to help the community by the end of the year.	**Generation:** powerful way to make the most of the data.The data is stored in a relational database, and the model is then used to create a model of a particular type of data
GPT2 SAE	**Generation:** very important part of the design process.The first step is to create a model that is easy to use and easy for the user to understand.	**Generation:** good idea. It's a way of doing a series of a month and then you can go to the next level and you don't have to worry about the other thing. You can do a two-year-old	**Generation:** great way to get started with your Android device. The first thing you need to do is install the Android SDK. The SDK is located in your computer.

(b) New prompt, 'year/month' and 'Phone' concept

Input Prompt: Latent concept modeling is a

	No Steer	Steer concept 6781: politic/law related	Steer concept 8984: aggressive tone
CoCoMix (ours)	**Generation:** technique that allows you to model a complex system in a way that is easy to understand and easy for the user to manipulate.	**Generation:** high-level policy: The White House does not have a policy on *<REMOVED>*'s plans. The White house said that *<REMOVED>* and his team had not discussed the matter with	**Generation:** thing? I'm not sure what is going on here.
GPT2 SAE	**Generation:** very important part of the design process.The first step is to create a model that is easy to use and easy for the user to understand.	**Generation:** common practice in the world of computer science. The new law, which was introduced in 2010, requires that all data in public use be treated as such.	**Generation:** thing of the past. And that's what we're talking about here. A concept?

(c) New prompt, 'Politic/law' and 'aggressive tone' concept

Figure 10: More qualitative demonstration of the concept steering effect. CoCoMix and GPT2 models are 350M and 124M parameter transformers, respectively. For CoCoMix, we manipulate the predicted logit $\mathbf{z}$, while for GPT2, we adjust the SAE concept space $\mathbf{c}$ by increasing the activation of a specific concept index

To further analyze the steerability enabled by CoCoMix, we conducted experiments using both the same prompt as in the main figure and a new prompt (in Figure 10). For consistency, we first applied steering on additional concepts identified during our analysis—"$ dollar" and "Phone"—using the same prompt as in Figure 5. These experiments confirmed that the model could effectively modulate its output based on these newly identified concepts, producing coherent and concept-aligned generations. Next, to verify whether the identified concepts generalize to different contexts, we experimented with a new prompt: *"Latent concept modeling is a"* and steered the model using the previously identified concepts "month/year" and "Phone." The results showed that the model successfully reproduced outputs aligned with these concepts, further supporting the robustness of our method. Additionally, we explored whether new concepts could be identified and steered using the same prompt. In this case, we identified two new concepts: "politics/law" and "aggressive tone." Steering the model with these new concepts demonstrated that the outputs could be effectively controlled to exhibit characteristics aligned with the corresponding concepts. These findings further highlight the flexibility and interpretability of our approach.

C Future Work and Societal Impact

Societal impact. We introduce a novel LLM pretraining framework that augments continuous concepts for the next token prediction. This framework not only improves the performance but also enhances the interpretability and controllability of the LLM. By enabling direct probing and analyzing the CoCoMix's concept space, one can achieve transparency in understanding and guiding model behavior. Additionally, the framework offers the potential to selectively exclude harmful or undesirable concepts during LLM pretraining, contributing to the development of more responsible and ethical AI systems. We believe this research provides new perspectives on language modeling through enhanced interpretability.

Future work and limitations. Several recent approaches, including ours, leverage additional compute to support more advanced forms of reasoning—such as test-time optimization or latent thinking via pause tokens. While this often leads to improved performance, it naturally increases computational cost. As shown in Section 3.3, CoCoMix achieves favorable compute efficiency, demonstrating stronger performance per FLOP compared to prior latent thinking methods. This opens up an interesting direction for future work: designing more informative and structured inputs that better utilize the compute budget of the Transformer to improve the performance. Also, future work could explore learning continuous concepts during pretraining without the need for distillation.