Next-Latent Prediction Transformers Learn Compact World Models

Jayden Teoh* **Manan Tomar** **Kwangjun Ahn** **Edward S. Hu** **Tim Pearce**
Pratyusha Sharma **Akshay Krishnamurthy** **Riashat Islam** **Alex Lamb**
John Langford

Microsoft Research

Abstract

Transformers replace recurrence with a memory that grows with sequence length and self-attention that enables ad-hoc lookups over past tokens. Consequently, they lack an incentive to compress history into compact latent states with consistent transition rules. This often leads to learning solutions that generalize poorly. We introduce **Next-Latent Prediction (NextLat)**, which extends standard next-token training with *self-supervised* predictions in the latent space. Specifically, NextLat trains a transformer to learn latent representations that are predictive of the next latent state given the next token. Theoretically, we show that these latents converge towards *belief states*, compressed information of the history necessary to predict the future. This simple auxiliary objective injects a recurrent inductive bias into transformers while leaving their architecture, parallel training, and inference unchanged. NextLat effectively encourages transformers to form compact internal world models with coherent belief states and transition dynamics—crucial properties not guaranteed by standard next-token prediction alone. Empirically, across benchmarks in world modeling, reasoning, planning, and language modeling, NextLat demonstrates significant gains over standard next-token prediction and other baselines in downstream accuracy, representation compression, and lookahead planning. Furthermore, NextLat enables *variable-length self-speculative decoding*, accelerating inference by up to $3.3\times$ in language modeling. NextLat stands as a simple and efficient paradigm for shaping transformer representations toward stronger generalization. Our code is available at `https://github.com/JaydenTeoh/NextLat`.

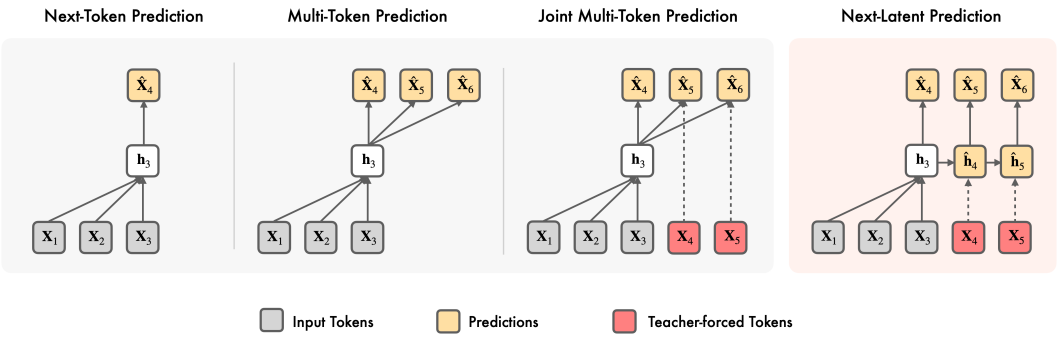


Figure 1: **Illustration of different predictive mechanisms.** Other methods supervise only in token space, leaving intermediate latent representations implicit. In contrast, NextLat explicitly trains the model to predict the next latent state (h_{t+1}) given the current latent state (h_t) and next token (X_{t+1}).

*Correspondence to: jayden_t@mit.edu

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

Ptolemy's geocentric model could accurately predict observations of the solar system from Earth's viewpoint, yet it was structurally convoluted: at times implying that the Moon came twice as close to Earth as other times. It was later supplanted by Copernicus's simpler, more compact heliocentric model, which provided accurate predictions that generalized beyond Earth's perspective. In learning theory, it is well known that simpler explanations of observations tend to generalize better [9, 48].

Modern transformers [89] stand in contrast to this principle. By replacing recurrence with a memory that scales with sequence length and self-attention that enables flexible look ups over past tokens, they achieve exceptional parallelization and predictive power. Yet, this very capability removes any inherent pressure to compress history into compact latent representations with consistent update rules. As such, transformers often learn task-specific shortcuts that fit the training data well but generalize poorly, especially in long-horizon planning tasks [4, 16, 56, 91]. How can we encourage transformers to form simpler, more principled explanations that avoid such shortcuts? A natural approach is to reinstate a key property of recurrent models: the ability to learn *compact world models* that channel future prediction through compressed representations of the past. We will show that this inductive bias can be introduced while also retaining the parallel training efficiency of transformers.

We introduce **Next-Latent Prediction (NextLat)**, which extends standard next-token prediction objective with self-supervised predictions in the latent space. NextLat jointly trains a transformer's latent representation alongside a latent dynamics model: the transformer encodes the history into compact latent states such that, when conditioned on the current latent and the next token (i.e., the "action"), the dynamics model can predict the transformer's next latent state. This drives the transformer to form compact latent summaries of past information with recurrent-like dynamics. In effect, NextLat enables the transformer to learn a compact world model while avoiding the sequential processing overhead of recurrent architectures. Importantly, NextLat leaves the transformer's architecture, inference, and parallel training procedure unchanged, introducing only a lightweight auxiliary loss on its latent representations during training. Our approach is inspired by the *self-predictive learning* paradigm in reinforcement learning (RL), a family of algorithms that learn representations by minimizing the prediction error of their own future latent states [85, 67]. An additional benefit of learning a latent dynamics model is that we can recursively compose predictions of future tokens directly in latent space without invoking the main transformer. This allows efficient multi-step lookahead using the lightweight dynamics model and supports *variable-length self-speculative decoding*.

The core contributions of this paper are threefold. First, we establish a theoretical foundation showing that NextLat shapes transformer representations into *belief states*—compact summaries of past information sufficient for predicting future observations—a property enabling planning and generalization, yet not guaranteed to emerge from next-token prediction alone. Second, we present a practical implementation of NextLat that preserves the transformer's architecture, inference procedure, and parallel training efficiency. By minimizing next-latent prediction error during training, we impose a structural preference that promotes compression and generalization, without adding parameters or computation at inference time. Finally, we empirically demonstrate NextLat's effectiveness across diverse domains spanning world modeling, reasoning, planning, and language modeling. Our results show that transformers trained with NextLat outperform standard next-token training and other baselines in representation compactness, lookahead planning, and downstream accuracy. In the language domain, NextLat achieves up to $3.3\times$ faster inference via variable-length self-speculative decoding. Together, these results position NextLat as an efficient and broadly applicable framework for learning compact, predictive, and generalizable representations in transformers.

2 Related Work

We are motivated by a long line of prior works in representation learning for prediction and control; a brief summary is provided here, with an extended discussion deferred to Section C.

Self-Supervised Learning. Self-supervised learning (SSL) is a framework for learning from unlabeled data, where a model generates its own supervisory signals from the structure of raw inputs. Across modalities such as vision, audio, and time series, SSL has proven highly effective for pretraining useful features, enabling downstream transfer that rivals, or even surpasses, models trained on labeled data [58, 7, 98]. However, latent-space SSL remains underexplored in language modeling.

NextLat falls under the *self-predictive representation learning* approach to SSL, which involves the joint learning of latent representations and a transition function that models how these representations evolve over a sequence. Self-prediction has driven state-of-the-art advances in RL [27, 96, 94, 80, 37].

Belief States. In both sequence modeling and RL, models must reason over long histories of observations. To mitigate this curse of dimensionality, prior work compresses history into latent representations that capture all information necessary for future prediction. In RL, this latent summary is formalized by Kaelbling et al. [46] as a *belief state*: *"a sufficient statistic for the past history ... no additional data about its past actions or observations would supply any further information about the current state of the world"*. While recurrent neural networks naturally enforce such compression, transformers have no such constraint—their internal state, or memory, grows linearly with sequence length. Recently, Belief State Transformers (BST; [41]) extended the notion of belief states to transformers, and demonstrated benefits in planning tasks. In contrast, NextLat learns belief states without requiring a separate transformer, and is significantly more computationally efficient.

World Models. Loosely, a world model is a predictive model of how the world works. Whether transformer language models implicitly learn world models remains under debate; some studies report emergent world understanding [71, 50, 32], while others find incoherent world structure [87, 88]. Successful world model approaches such as *MuZero* [79], *Dreamer* [34–36], and *Genie* [10] share a common principle: they explicitly learn a latent dynamics that predict future states from current states (encodings of past observations) and actions. Yet, learning such latent dynamics for autoregressive language modeling in transformers remains underexplored. NextLat addresses this gap by introducing a self-prediction objective that explicitly learns a latent dynamics model which governs how the transformer's latent states evolve given new tokens (i.e., "actions"), thereby enabling the transformer to learn compact latent abstractions of the world with consistent dynamics.

Beyond Next-Token Prediction. A growing body of work has highlighted the myopic nature of the next token prediction, which limits the model's capability in downstream tasks such as planning and reasoning [6, 66]. Recent works have also found improvements from richer supervision signals that predict further into the future [28, 55, 41, 2]. However, these approaches operate predominantly in the token space. NextLat takes a different approach: it shifts prediction into the latent space, enforcing self-consistent dynamics over the model's latent representations rather than its token outputs.

3 Methodology: Next-Latent Prediction

In this section, we introduce a simple, yet powerful, method for learning belief states in transformers via next-latent prediction (or more specifically, via *next-hidden state prediction*[1]). We begin by defining belief states in sequence modeling.

Definition 3.1 (Belief states in sequence modeling)**.** Let $X_{1:T}$ denote a token sequence $X_1, \ldots, X_T$. A random variable $\mathbf{b}_t = g(X_{1:t})$ is a *belief state* for the history $X_{1:t}$ if, for every bounded measurable function f of the future,

$$\mathbb{E}[f(X_{t+1:T}) \mid \mathbf{b}_t] = \mathbb{E}[f(X_{t+1:T}) \mid X_{1:t}] \quad \text{a.s.}$$

Equivalently, $\mathbf{b}_t$ is a *sufficient statistic* [81] of the history $X_{1:t}$ for predicting the future tokens, i.e., from which we can sample from the distribution $\mathbb{P}(X_{t+1:T} \mid X_{1:t})$. Next, we describe how next-latent prediction enables transformers to learn belief states and improves data efficiency.

3.1 Why Next-Latent Prediction?

Here we analyze an idealized next-latent prediction transformer which successfully optimizes both next-token prediction and next-latent prediction with respect to an underlying data distribution.

Theorem 3.2. *Consider the joint learning of three components:*

1. *a transformer with parameters θ that produces hidden states $\mathbf{h}_t$ at each time step t,*
2. *an output head p_θ modeling the next-token distribution, and*
3. *a latent dynamics model p_ψ modeling the transition dynamics of the transformer's hidden states.*

[1] In the sequence modeling literature, intermediate latent representations are often referred to as "hidden states". To disambiguate, we use the term "latent state" to broadly refer to learned representations within the transformer's residual stream, and "hidden state" to refer to a subset of this representation—specifically, the final layer's output (i.e., the pre-logit activations).

If NextLat successfully optimizes the following objectives:

$$\textbf{(Next-Token Consistency):} \quad p_\theta(X_{t+1} \mid \mathbf{h}_t) = \mathbb{P}(X_{t+1} \mid X_{1:t}), \tag{1}$$

$$\textbf{(Transition Consistency):} \quad p_\psi(\mathbf{h}_{t+1} \mid \mathbf{h}_t, X_{t+1}) = \mathbb{P}(\mathbf{h}_{t+1} \mid X_{1:t+1}), \tag{2}$$

then $\mathbf{h}_t$ must be a belief state for the sequence $X_{1:t}$. Note that the right-hand side of Equation 2 is the transition law induced by the transformer's weights[2].

Proof Sketch. A formal proof by backward induction is provided in Section E. Intuitively, optimizing for next-token (Equation 1) and transition (Equation 2) consistency ensures existence of measurable maps, i.e., p_θ and p_ψ, that allow recursive decoding of future tokens from $\mathbf{h}_t$:

$$\mathbf{h}_t \xrightarrow[\text{decode token}]{p_\theta} X_{t+1} \xrightarrow[\text{update state}]{p_\psi} \mathbf{h}_{t+1} \xrightarrow[\text{decode token}]{p_\theta} X_{t+2} \xrightarrow[\text{update state}]{p_\psi} \mathbf{h}_{t+2} \cdots \xrightarrow{p_\theta} X_T.$$

For these maps to exist, and be learned, $\mathbf{h}_t$ must jointly optimize toward a belief state—a sufficient statistic for the history to predict the future.

Remark. Optimizing only next-token consistency (i.e., Equation 1) in standard autoregressive transformers does not guarantee that $\mathbf{h}_t$ forms a belief state (see Theorem 3 in Hu et al. [41]). Intuitively, self-attention enables ad-hoc lookup of past tokens, so there is no pressure to compress all necessary information about the past into compact latent summaries at every time step.

NextLat Enables Better Data Efficiency. In prior methods that supervise in token space (see Figure 1), the learning signal is only anchored to the next token, or multiple tokens. NextLat additionally supervises in latent space. Specifically, the model is trained to predict its own next hidden state $\mathbf{h}_{t+1}$, which parameterizes the full predictive distribution over X_{t+2}. This shifts supervision from individual one-hot token labels to distribution-level alignment. Moreover, because the latent dynamics compose recursively—each latent is trained to predict the next—$\mathbf{h}_{t+1}$ implicitly carries information about future states $\mathbf{h}_{t+2}, \mathbf{h}_{t+3}, \ldots$. As a result, NextLat not only provides learning signals that are dense in the vocabulary space, but also propagates information about future tokens into earlier representations.

3.2 Learning to Predict Next-Latent States

We now describe the practical implementation of NextLat. Our NextLat implementation operates primarily on the hidden states (i.e., the final-layer outputs) as they provide compact, fixed-dimensional vectors through which gradients can be propagated through the entire transformer efficiently. As usual, we optimize the model for next-token prediction (Equation 1) using the cross-entropy loss:

$$\mathcal{L}_{\text{next-token}}(\theta) = \mathbb{E}_{t<T}\big[-\log p_\theta(X_{t+1} \mid \mathbf{h}_t)\big].$$

NextLat additionally enforces transition consistency (Equation 2) of the hidden states by introducing a latent dynamics model p_ψ that predicts the next hidden state $\mathbf{h}_{t+1}$ directly from $(\mathbf{h}_t, X_{t+1})$. For a deterministic transformer model, $\mathbb{P}(\mathbf{h}_{t+1} \mid \mathbf{h}_t, X_{t+1})$ is a Dirac distribution, and we can optimize p_ψ via regression[3]. Moreover, observe that an ideal latent dynamics model should admit recursive consistency: its one-step map should compose correctly across multiple steps. Let $\hat{\mathbf{h}}_{t+d} = p_\psi(\mathbf{h}_t, X_{t+1:t+d})$ denote the recursive rollout of p_ψ over a d-step horizon using teacher-forced tokens $X_{t+1:t+d}$. We supervise all d intermediate rollouts using the smooth L1 loss:

$$\mathcal{L}_{\text{next-h}}(\theta, \psi; d) = \mathbb{E}_t\Big[\frac{1}{d}\sum_{i=1}^{d} \text{SmoothL1Loss}\big(\mathbf{sg}[\mathbf{h}_{t+i}], \hat{\mathbf{h}}_{t+i}\big)\Big], \tag{3}$$

where $\mathbf{sg}[\cdot]$ denotes the stop-gradient operator, used to prevent representational collapse in self-predictive learning [67]. **Note that belief state convergence (i.e., Theorem 3.2) already holds for $d = 1$. Multi-step supervision serves only to provide richer learning signal.** To further align the semantics of predicted states $\hat{\mathbf{h}}$ with true states, we introduce a complementary KL objective enforcing agreement in token-prediction space:

$$\mathcal{L}_{\text{KL}}(\theta, \psi; d) = \mathbb{E}_t\Big[\frac{1}{d}\sum_{i=1}^{d} D_{\text{KL}}\Big(p_\theta^{\mathbf{sg}}(\cdot \mid \mathbf{sg}[\mathbf{h}_{t+i}]) \,\|\, p_\theta^{\mathbf{sg}}(\cdot \mid \hat{\mathbf{h}}_{t+i})\Big)\Big], \tag{4}$$

[2]We adopt a probabilistic formulation to retain generality with respect to stochastic transformer models, e.g., Fleuret [20].

[3]If considering a stochastic transformer model, p_ψ can be optimized through variational inference.

where the output head $p_\theta^{\text{sg}}(\cdot)$ is frozen so that gradients flow only through the latent dynamics model. This KL acts similarly to *knowledge distillation* [40], providing soft supervision that guides learning of p_ψ. It also resembles *observation reconstruction* in self-predictive RL [83, 67], encouraging $\hat{\mathbf{h}}_{t+i}$ to reproduce the distribution over next observations (i.e., the output head's logits).

Overall Objective. The final NextLat objective combines all components, minimizing the loss:

$$\mathcal{L}_{\text{NextLat}}(\theta, \psi; d, \lambda_{\text{next-h}}, \lambda_{\text{KL}}) = \mathcal{L}_{\text{next-token}}(\theta) + \lambda_{\text{next-h}}\, \mathcal{L}_{\text{next-h}}(\theta, \psi; d) + \lambda_{\text{KL}}\, \mathcal{L}_{\text{KL}}(\theta, \psi; d), \quad (5)$$

where $\lambda_{\text{next-h}}, \lambda_{\text{KL}} > 0$ are scalar coefficients. During inference, the learned transformer decodes autoregressively as usual; p_ψ is used only during training to shape the transformer representations.

In the experiments that follow, we parameterize p_ψ using simple MLPs to show that NextLat yields significant performance gains over baselines even without sophisticated latent dynamics architectures. Implementation details of the NextLat algorithm can be found in Section F. To illustrate the simplicity of our approach, we also include a PyTorch-style pseudocode of the NextLat objective in Algorithm 1.

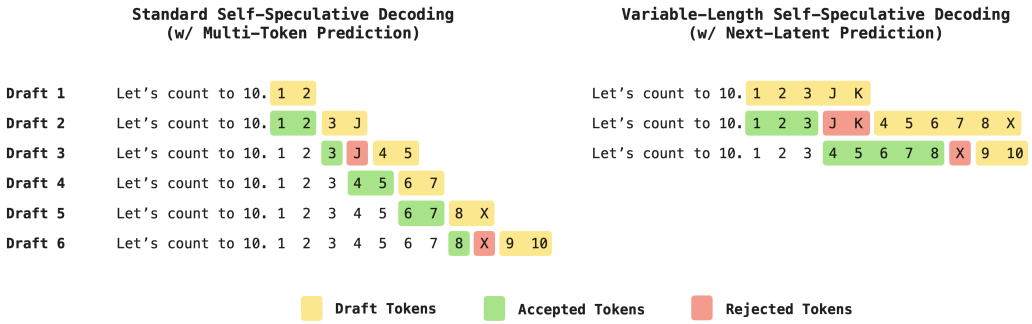


Figure 2: Comparison of self-speculative decoding with multi-token prediction (MTP) vs. NextLat.

Variable-Length Self-Speculative Decoding. Figure 2 compares self-speculative decoding in MTP and NextLat. Standard MTP models are trained to predict the next d tokens and typically can only draft up to d tokens per cycle (e.g., $d = 2$ tokens in this figure). In contrast, NextLat can recursively compose predictions via its latent dynamics: $\mathbf{h}_t \to X_{t+1} \to \mathbf{h}_{t+1} \to X_{t+2} \to \cdots$, enabling variable-length speculative decoding and reducing draft-verification cycles. We show later in Section 4.4 that even with shallow training horizons ($d = 1, 2$), NextLat maintains coherent dynamics beyond the training horizon, supporting longer drafts and faster inference than MTP baselines.

4 Experiments

We evaluate NextLat on four key axes where modeling coherent latent dynamics and beliefs matter: world modeling, reasoning, planning, and language modeling. Our baselines include transformer-based belief-learning methods, i.e., BST [41] and JTP [2], as well as standard next-token prediction (GPT) and multi-token prediction (MTP; [28]). Hereafter, we use the term "horizon" to refer to the multi-step prediction horizon d in JTP, MTP, and NextLat, and we match horizon across these methods in all experiments to ensure fair comparisons. Additional experiment details such as setup, hyperparameters, evaluation procedure, etc., are in Section I.

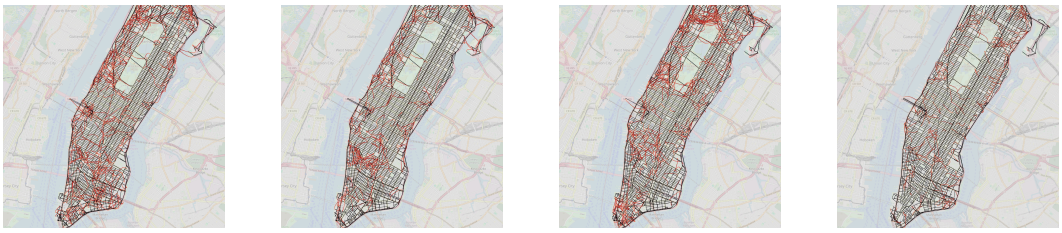
4.1 World Modeling

Table 1: Comparison of models trained on Manhattan taxi rides against the true world model.

	Next-Token Test (↑)	Valid Trajectories (↑)	Sequence Compression (↑)	Effective Latent Rank (↓)	Detour Robustness (↑)
GPT	100%	97.0%	0.65	160.1	85.0%
MTP	100%	98.1%	0.64	57.7	**95.0%**
JTP	100%	97.1%	0.32	215.8	87.0%
NextLat	100%	**98.7%**	**0.71**	**52.7**	**95.0%**
True world model	100%	100%	1.00	—	100%

Vafa et al. [87] revealed that transformers trained on turn-by-turn taxi ride trajectories in Manhattan can achieve near-perfect next-token accuracy, yet their internal maps remain incoherent; they reconstruct streets with impossible orientations and even flyovers above other roads. We train the models using the *random walks* dataset from Vafa et al. [87], which consists of random traversals between taxi pickup and dropoff points in Manhattan. We exclude BST due to its high computational cost. For JTP, MTP, and NextLat, we set the multi-step prediction horizon at $d = 8$. We evaluate world-modeling performance using five comprehensive metrics:

1. Next-Token Test: Percentage of top-1 token predictions that are legal turns under teacher-forcing on in-distribution validation sequences.
2. Valid Trajectories: Percentage of valid traversals for OOD origin–destination pairs.
3. Sequence Compression: Percentage of cases where the model produces identical continuations for two different traversals arriving at the same state and sharing the same destination.
4. Effective Latent Rank: Effective dimension of hidden states measured as the Shannon entropy of normalized singular values [75]; lower values indicate better compression.
5. Detour Robustness: Percentage of valid traversals for OOD origin–destination pairs when we substitute the model's top-1 prediction with random detours 75% of the time.



(a) GPT (b) MTP (c) JTP (d) NextLat (ours)

Figure 3: Reconstructed maps from transformers trained on Manhattan taxi rides.

Results. As expected, all models achieved 100% accuracy on the next-token test. However, next-token accuracy cannot meaningfully assess the quality of a model's learned world model. In Figure 3, we visualize each model's internal map using the reconstruction algorithm proposed by Vafa et al. [87]. Visibly, the transformer trained with NextLat exhibits an internal map most consistent with the true world model. Although not perfect, its inconsistencies (red edges) are sparse and mostly local. Beyond this qualitative evidence, NextLat consistently outperforms all baselines on all metrics. On the trajectory validity and detour robustness metrics, NextLat demonstrates the **strongest generalization to OOD pickup–dropoff pairs**, even when random detours are introduced.

Next, we analyze the compactness of the learned world models. A model that accurately captures the underlying states and transitions should assign identical continuations to trajectories that end in the same state (i.e., intersection in Manhattan). By this criterion, NextLat achieves the **highest sequence compression of 0.71**. The Manhattan graph comprises only 4,580 intersections and 9,846 edges, and therefore an effective world model should require only a modest latent dimensionality. Indeed, NextLat has the **lowest effective latent rank of 52.7—over 3x smaller than GPT's**. These results suggest that NextLat enables transformers to learn more efficient world models—ones that are both accurate in their predictive structure and compact in their internal representation of the environment.

4.2 Reasoning

Countdown [14] is a challenging mathematical reasoning task which even frontier models such as GPT-4 [1] only achieve 4% by default [92]. The goal of the task is to combine a set of given numbers with basic arithmetic operations $(+, -, \times, \div)$ to obtain a target number. For e.g., given the numbers $\{90, 8, 20, 50\}$, the target number 24 can be obtained using the following solution: $90 \times 8 = 720$, $50 - 20 = 30$, $720 \div 30 = 24$. Countdown poses a difficult combinatorial search problem due to its large branching factor and the need to efficiently explore the solution space to reach the target number.

Figure 4: Performance on Countdown. Best result is **bolded**, and second best is underlined.

Model	Accuracy (%)		
	d=1	d=4	d=8
GPT	------	33.1	------
BST	------	42.3	------
MTP	39.2	49.7	57.3
JTP	39.0	49.4	55.0
NextLat	54.8	<u>57.6</u>	**58.7**

Results. As shown in Figure 4, NextLat consistently outperforms all baselines on the Countdown task. Notably, even when trained with a shallow prediction horizon of $d = 1$, it substantially surpasses the MTP and JTP baselines with the same horizon (**>38% improvement**). To understand this gap, we analyzed the equations generated by each model and their validity, i.e., whether the left-hand side equals the right-hand side. As shown in Figure 5, most errors occur in the final equation, reflecting a lack of planning: models only realize failure at the end and compensate with an invalid final step—a behavior termed *the regretful compromise* [93]. NextLat exhibits **stronger lookahead planning**: it achieves substantially higher mean validity in the final equation (54.8%) compared to the next best baseline (42.3%). This suggests that its latent prediction objective enables it to anticipate long-range dependencies and form globally consistent plans, reducing myopic errors. NextLat achieves the leading performance with horizons $d = 4$ and 8.

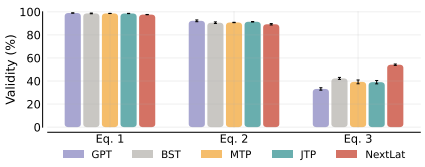


Figure 5: Validity of equations generated by $d = 1$ models.

4.3 Planning

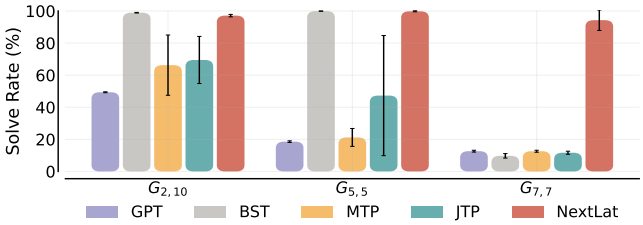


Figure 6: Accuracy on Path-Star graph task.

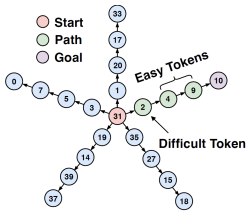


Figure 7: A $G_{5,5}$ Path-Star graph [6].

The Path-Star benchmark is a minimal lookahead planning task that exposes the myopic behavior of next-token prediction. Next-token prediction exploits local shortcuts and fails to solve the task despite its simplicity. A Path-Star graph [6] $G_{k,\ell}$ consists of a center node and k disjoint arms, each with $\ell - 1$ nodes. Each training sequence consists of the edge list, the start and end nodes, and the correct path from start to end. For MTP, JTP, and NextLat, we set the multi-step prediction horizon to $d = \ell - 2$ so that the end node lies within the multi-step prediction horizon from the center node. We evaluate performance across three graph configurations: $G_{2,10}$, $G_{5,5}$, and $G_{7,7}$.

Results. Methods such as BST, MTP, and JTP attempt to mitigate shortcut learning through multi-token predictions. However, these approaches still operate in token space and remain susceptible to local n-gram regularities. Indeed, as shown in Figure 6, these methods fail to solve the task consistently. BST degrades significantly on the larger graph $G_{7,7}$. In contrast, NextLat achieves **near-100% solve rate across all configurations**. This suggests that predicting in latent space instead of token space, and enforcing consistent transition dynamics at the representation level, better avoids shortcut learning. Furthermore, given the limited training samples (i.e., 200k) in this task, NextLat's strong performance suggests that it is more data efficient, consistent with our analysis in Section 3.1.

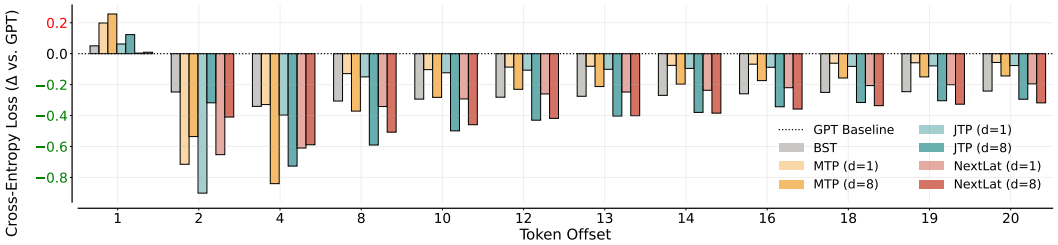


Figure 8: Cross-entropy loss difference relative to GPT, measured by linear probes trained on frozen hidden states to predict tokens at varying token offsets. Lower values indicate better predictivity.

Next, we evaluate models on TinyStories [18], a dataset of synthetic short stories. Storytelling is inherently a long-horizon planning task; a coherent narrative requires maintaining persistent entities,

tracking causal relationships, and delayed resolutions across many timesteps. After pretraining on TinyStories, we freeze the models' parameters and train 20 linear probes to independently predict tokens at offsets $1, \ldots, 20$ steps ahead from the hidden states of the frozen models using the same dataset. This tests whether the models' representations encode information predictive of future tokens, or just local token correlations. We compare MTP, JTP, and NextLat trained with $d = 1, 8$.

Results. Figure 8 shows the probe performance of the models relative to probes trained on GPT's hidden states. For clarity, we display only selected token offsets here (see Figure 17 for full results). Observe that the additional token-level objectives in BST, MTP, and JTP consistently cause degradation in next-token prediction (i.e., token offset = 1). Moreover, probe performance on JTP and MTP representations declines sharply with increasing token offset. This indicates that these multi-step prediction models, lacking guarantees of learning belief states, could encode information useful only for short-horizon prediction. In contrast, NextLat matches GPT on next-token performance and exhibits the strongest long-range predictive capability (up to 20 tokens ahead) for both $d = 1, 8$. This suggests that NextLat induces belief-like representations that encode predictive information about future events—lookahead planning essential for coherent long-range narrative generation.

4.4 Language Modeling

Table 2: Downstream language modeling evaluation on 1.3B-parameter models trained on 100B FineWeb-Edu tokens. Best scores are in bold and second-best are underlined.

Model	FW-Edu ppl ↓	Wiki. ppl ↓	LAMB. ppl ↓	LAMB. acc ↑	PIQA acc ↑	HellaS. acc ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc ↑	SIQA acc ↑	SciQ acc ↑	Avg.
GPT	**10.52**	**17.93**	20.26	42.07	73.45	**58.79**	60.46	68.18	39.16	42.32	86.10	58.82
JTP (d=1)	11.08	19.28	21.88	41.35	**74.92**	57.43	58.64	68.73	39.25	42.99	87.30	58.83
JTP (d=2)	11.18	19.60	22.11	41.37	73.34	56.84	59.98	68.86	38.57	**43.35**	86.70	58.63
MTP (d=1)	10.90	18.82	20.23	41.26	74.32	58.05	**60.54**	68.52	38.91	42.84	85.40	58.73
MTP (d=2)	11.00	18.61	18.34	43.43	72.80	57.92	59.35	68.35	39.08	41.97	86.60	58.69
NextLat (d=1)	10.83	18.39	19.77	41.08	73.07	58.35	59.27	69.65	39.68	43.24	86.00	58.79
NextLat (d=2)	10.88	18.44	**17.83**	**43.86**	73.61	57.79	59.20	**69.74**	**40.10**	41.91	**87.50**	**59.21**

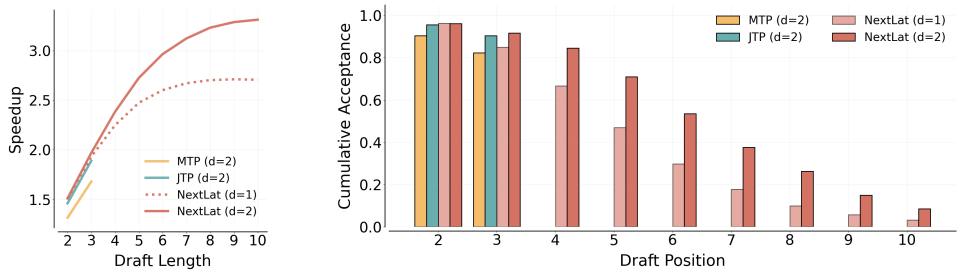
We pretrain 1.3B-parameter models on 100B tokens from the FineWeb-Edu dataset [72], excluding BST due to its high computational cost. We then evaluate the zero-shot accuracy of the models on multiple-choice benchmarks using the LM Evaluation Harness [26]. We further assess the self-speculative decoding performance of JTP, MTP, and NextLat across Wikipedia, Books, Code, and Math domains using the speculative sampling algorithm of Leviathan et al. [49]. We primarily focus our analysis on training with horizons of $d = 1$ and $d = 2$ to highlight the advantages of NextLat's variable-length self-speculative decoding, which enables drafting beyond the training horizon. To this end, we vary NextLat's speculative draft length between 2 and 10 tokens and report the highest inference speedup achieved for each domain.

Table 3: Speedup relative to standard autoregressive sampling from the base transformer and average accepted draft tokens per speculative step, measured on 8×NVIDIA B200 GPUs. Note that "Accepted Tokens" excludes the next-token prediction, which is always accepted.

	Wikipedia		Books		Code		Math	
Model	Speedup	Accepted Tokens	Speedup	Accepted Tokens	Speedup	Accepted Tokens	Speedup	Accepted Tokens
JTP (d=1)	1.46	0.96	1.47	0.97	1.47	0.98	1.46	0.97
JTP (d=2)	1.88	1.84	1.90	1.89	1.88	1.85	1.89	1.86
MTP (d=1)	1.38	0.91	1.39	0.95	1.40	0.97	1.39	0.95
MTP (d=2)	1.68	1.72	1.72	1.83	1.75	1.91	1.72	1.84
NextLat (d=1)	2.68	3.52	2.72	3.64	2.29	2.66	2.30	2.72
NextLat (d=2)	**3.21**	**4.59**	**3.32**	**4.86**	**2.38**	**2.83**	**2.87**	**3.94**

Results. Table 2 presents the language modeling perplexity (ppl) and zero-shot accuracy (acc) of the pretrained models. Consistent with Gloeckle et al. [28], we do not observe significant improvements in multiple-choice task accuracy over next-token training when using multi-token objectives. NextLat ($d = 2$) is the **only multi-step prediction objective that meaningfully improves average accuracy over GPT (59.21 vs. 58.82)**, but these improvements are modest and not consistent across tasks. Larger model sizes might be necessary to see more significant improvements. In contrast, MTP and JTP do not benefit zero-shot accuracy over next-token training. Notably, NextLat **better preserves next-token perplexity** than MTP and JTP across domains. Preserving high fidelity in next-token prediction is arguably important, as prior work shows that lower pretraining perplexity correlates with improved aggregate downstream and post-fine-tuning performance at larger model scales [24, 97].

Next, Table 3 shows the self-speculative decoding results. Observe that the average accepted tokens per drafting step for NextLat far exceeds its training horizons $d = 1, 2$, indicating that the learned latent dynamics remains coherent over extended rollouts and highlighting the long-range predictive capability of the induced belief state representations. Crucially, this ability to support variable-length self-speculative decoding enables NextLat to achieve **substantially higher inference speedups than MTP and JTP across all domains**. Figure 9 further shows speedup and cumulative acceptance rate versus draft length on the FineWeb-Edu validation set. As shown, there is a clear benefit from drafting tokens beyond the training horizon for NextLat: speedup continues to increase with draft length up to $3.3\times$, and fully valid drafts persist even up to length 10. In Section I.6.1, we extend our comparisons to include JTP and MTP trained with larger horizons ($d = 4$). Note that the training cost of JTP and MTP increases substantially with larger d (see Table 4), making JTP ($d = 4$) and MTP ($d = 4$) significantly more expensive to train than NextLat ($d = 1, 2$). Even with longer multi-token supervision, these baselines still fail to surpass NextLat in zero-shot accuracy and speculative decoding performance. This highlights a key advantage of NextLat's variable-length speculative decoding: it enables long speculative drafts while requiring training only at shallow, computationally efficient horizons.



(a) (b)

Figure 9: (a) Inference speedup of JTP, MTP, and NextLat on the FineWeb-Edu validation set. (b) Cumulative acceptance rate of draft tokens (product of per-position acceptance probabilities), indicating how likely the entire speculative draft remains valid as lookahead increases.

5 Discussions

In this section, we briefly compare NextLat's computational cost to baselines and discuss the use of NextLat to train recurrent neural networks (RNNs) without recurrence. A more detailed discussion, including comparisons with baselines and RNNs, is provided in Section B.

Table 4: Training speed, parameter count and gradient signals provided on FineWeb-Edu for $d = 1, 2, 8$. If only a single value is shown, it means that the values are the same for all horizons. All training speeds are measured on a single NVIDIA B200 GPU with batch size of 33k tokens.

	GPT	BST	MTP	JTP	NextLat
Training parameters ($d = 1, 2, 8$)	1.32B	2.57B	1.37B / 1.42B / 1.72B	1.34B	1.40B
Inference parameters	1.32B	1.32B / 2.57B	1.32B	1.34B	1.32B
Training steps/second ($d = 1, 2, 8$)	3.09	0.89	2.80 / 2.58 / 1.70	3.15 / 2.92 / 2.02	3.09 / 2.79 / 1.73
Gradients	$O(T)$	$O(T^2)$	$O(Td)$	$O(Td)$	$O(Td)$

Computational Costs. Table 4 summarizes the training/inference parameters, training speed, and gradient signals for each method. BST benefits from $O(T^2)$ supervision but training is extremely costly (over $3\times$ slower than NextLat). In contrast, NextLat with $d = 1$ incurs negligible overhead over GPT while achieving the same belief-state learning guarantees. MTP requires additional transformer layers as d increases, whereas JTP and NextLat keep parameter counts fixed. At $d = 1$, JTP and NextLat have similar training speeds[4] while MTP lags behind. At $d = 8$, JTP is faster because NextLat has to unroll its latent dynamics model sequentially to compute multi-step losses.

[4]JTP exhibiting a slightly faster training speed than GPT at $d = 1$ is likely an artifact of `torch.compile()` due to differences in the computation graph.

Nonetheless, this modest increase in computation for NextLat yields substantially better performance than JTP across all benchmarks. Importantly, NextLat's computation remains far more efficient than that of RNNs, which we discuss further in Section B.2.

Training RNNs without recurrence. NextLat ($d = 1$) can be viewed as *fully parallel* co-training of a transformer and a recurrent neural network (RNN). The transformer forecasts latent states across the sequence in parallel, while the latent dynamics model (the RNN) is trained using only one-step latent predictions. This co-training process does not require any sequential computation across time.

To study this interaction, we consider the A_5 word problem, a difficult *state-tracking* task involving even permutations of five elements. The problem is NC^1-complete and therefore inexpressible by constant depth, fixed-precision transformers, which lie in $\text{TC}^0 \subset \text{NC}^1$ [62][5]. We train 2-layer transformers using next-token prediction (GPT) and NextLat ($d = 1$) on the A_5 word problem with a sequence length of 12 tokens. We then test the transformers and the latent dynamics model (RNN) on their length generalization to sequences of length 36. At inference, the RNN only uses the transformer backbone to initialize the hidden state for the first token, after which all subsequent transitions and predictions come from the RNN alone. Importantly, the RNN also has far fewer parameters than the transformer (2.62M vs. 6.43M parameters).

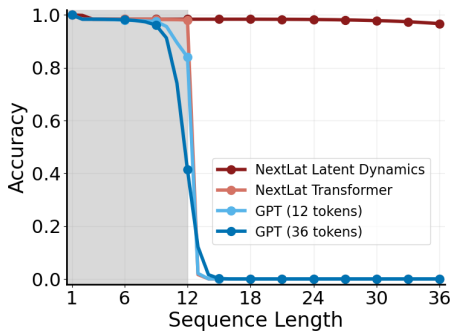


Figure 10: Length Generalization for A_5 word problem.

In Figure 10, the transformer trained with **NextLat exhibits better state-tracking performance than GPT** within the 12-token training horizon. From a circuit complexity perspective, solving problems in NC^1 requires transformers with $O(\log T)$ depth [62, 56]. This explains why the shallow 2-layer transformers (both GPT and NextLat) are unable to generalize beyond 12 tokens. Even GPT trained directly on 36 tokens fails, consistent with previous results [64]. Surprisingly, however, the latent dynamics model learned under NextLat generalizes to 36-token sequences ($> 95\%$ accuracy). Thus, **the co-trained RNN generalizes far beyond the expressivity limits of the underlying transformer, despite being trained entirely in parallel**. NextLat may therefore offer a mechanism for partially escaping the practical limitations imposed by the *parallelism tradeoff* [62]: the transformer provides efficient parallel representation learning, while the RNN focuses on learning generalizable sequential update rules.

6 Conclusion

In this paper, we introduced Next-Latent Prediction (NextLat), a simple yet powerful framework that augments next-token training with self-supervised latent-space prediction, enabling transformers to learn belief-state representations. Theoretically, we show that the pressure to form concise latent summaries of past information sufficient to predict the future arises naturally from the NextLat objective. Empirically, NextLat yields more compact, predictive, and generalizable representations across tasks in world modeling, reasoning, planning, and language modeling—all without changing the transformer's architecture, parallel training efficiency, or inference procedure. By reintroducing a recurrent inductive bias through self-predictive latent dynamics, NextLat unifies the inherent bias toward compact and temporally consistent representations of recurrent models with the scalability and parallelism of transformers. Our method is broadly applicable for training transformers in autoregressive sequence modeling domains. Looking ahead, we view NextLat as a step toward training objectives that endow autoregressive sequence models with simpler, more compact, and therefore more generalizable representations of complex systems. We refer motivated readers to the appendix for further details, including limitations and future work, extended discussions and related work, background on belief states, implementation details, ablation studies, and additional experimental details.

[5]This assumes the widely held conjecture that $\text{TC}^0 \neq \text{NC}^1$.

Acknowledgments and Disclosure of Funding

We thank Andrea Zanette, Jason Eisner, Jordan T. Ash, Khai Loong Aw, Michael C. Mozer, Pradeep Varakantham, Ying Fan, and the MSR AI Frontiers team for their valuable discussions and support.

Funding. This work was funded by Microsoft Research and conducted while J.T. was an intern at Microsoft Research.

Competing interests. The authors declare no competing interests beyond their current or former employment at Microsoft.

References

[1] 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.

[2] Kwangjun Ahn, Alex Lamb, and John Langford. Efficient joint prediction of multiple future tokens, 2025. URL https://arxiv.org/abs/2503.21801.

[3] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. Smollm2: When smol goes big – data-centric training of a small language model, 2025. URL https://arxiv.org/abs/2502.02737.

[4] Cem Anil, Yuhuai Wu, Anders Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 38546–38556. Curran Associates, Inc., 2022.

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

[6] Gregor Bachmann and Vaishnavh Nagarajan. The pitfalls of next-token prediction. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 2296–2318. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/bachmann24a.html.

[7] Randall Balestriero, Mark Ibrahim, Vlad Sobal, Ari Morcos, Shashank Shekhar, Tom Goldstein, Florian Bordes, Adrien Bardes, Gregoire Mialon, Yuandong Tian, Avi Schwarzschild, Andrew Gordon Wilson, Jonas Geiping, Quentin Garrido, Pierre Fernandez, Amir Bar, Hamed Pirsiavash, Yann LeCun, and Micah Goldblum. A cookbook of self-supervised learning, 2023. URL https://arxiv.org/abs/2304.12210.

[8] 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.

[9] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred Warmuth. Classifying learnable geometric concepts with the vapnik-chervonenkis dimension. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 273–282, 1986.

[10] Jake Bruce, Michael Dennis, Ashley Edwards, Jack Parker-Holder, Yuge (Jimmy) Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, Yusuf Aytar, Sarah Bechtle, Feryal Behbahani, Stephanie Chan, Nicolas Heess, Lucy Gonzalez, Simon Osindero, Sherjil Ozair, Scott Reed, Jingwei Zhang, Konrad Zolna, Jeff Clune, Nando De Freitas, Satinder Singh, and Tim Rocktäschel. Genie: generative interactive environments. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[11] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL https://arxiv.org/abs/2302.01318.

[12] 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.

[13] Roger C Conant and W Ross Ashby. Every good regulator of a system must be a model of that system. *International journal of systems science*, 1(2):89–97, 1970.

[14] Countdown. Countdown (game show), 2025. URL https://en.wikipedia.org/wiki/Countdown_(game_show). [Online; accessed 12-October-2025].

[15] Kenneth James Williams Craik. *The nature of explanation*, volume 445. CUP Archive, 1967.

[16] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Fkckkr3ya8.

[17] Ezra Edelman, Nikolaos Tsilivis, Benjamin L. Edelman, Eran Malach, and Surbhi Goel. The evolution of statistical induction heads: In-context learning markov chains. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 64273–64311. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/75b0edb869e2cd509d64d0e8ff446bc1-Paper-Conference.pdf.

[18] Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.

[19] Hugging Face. Dataset card for bookcorpusopen, 2021. URL https://huggingface.co/datasets/bookcorpusopen.

[20] François Fleuret. The free transformer, 2025. URL https://arxiv.org/abs/2510.17558.

[21] Wikimedia Foundation. Wikimedia downloads. URL https://dumps.wikimedia.org.

[22] Bruce A Francis and Walter Murray Wonham. The internal model principle of control theory. *Automatica*, 12(5):457–465, 1976.

[23] Karl Friston. The free-energy principle: a unified brain theory? *Nature reviews neuroscience*, 11(2):127–138, 2010.

[24] Samir Yitzhak Gadre, Georgios Smyrnis, Vaishaal Shankar, Suchin Gururangan, Mitchell Wortsman, Rulin Shao, Jean Mercat, Alex Fang, Jeffrey Li, Sedrick Keh, Rui Xin, Marianna Nezhurina, Igor Vasiljevic, Jenia Jitsev, Luca Soldaini, Alexandros G. Dimakis, Gabriel Ilharco, Pang Wei Koh, Shuran Song, Thomas Kollar, Yair Carmon, Achal Dave, Reinhard Heckel, Niklas Muennighoff, and Ludwig Schmidt. Language models scale reliably with over-training and on downstream tasks, 2024. URL https://arxiv.org/abs/2403.08540.

[25] Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D Goodman. Stream of search (sos): Learning to search in language. *arXiv preprint arXiv:2404.03683*, 2024.

[26] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

[27] Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G. Bellemare. DeepMDP: Learning continuous latent space models for representation learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2170–2179. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/gelada19a.html.

[28] Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[29] Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. *arXiv preprint arXiv:2310.02226*, 2023.

[30] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024. URL https://arxiv.org/abs/2312.00752.

[31] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces, 2022. URL https://arxiv.org/abs/2111.00396.

[32] Wes Gurnee and Max Tegmark. Language models represent space and time. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=jE8xbmvFin.

[33] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2(3), 2018.

[34] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[35] Danijar Hafner, Timothy P. Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=0oabwyZbOu.

[36] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. 640(8059):647–653, 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-08744-2. URL https://doi.org/10.1038/s41586-025-08744-2.

[37] Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control, 2022. URL https://arxiv.org/abs/2203.04955.

[38] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023. URL https://arxiv.org/abs/1606.08415.

[39] Vincent Herrmann, Róbert Csordás, and Jürgen Schmidhuber. Measuring in-context computation complexity via hidden state prediction. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org, 2025.

[40] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[41] Edward S. Hu, Kwangjun Ahn, Qinghua Liu, Haoran Xu, Manan Tomar, Ada Langford, Jayden Teoh, Bryon Xu, David Yan, Dinesh Jayaraman, Alex Lamb, and John Langford. The belief state transformer, 2025. URL https://arxiv.org/abs/2410.23506.

[42] 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.

[43] Hai Huang, Yann LeCun, and Randall Balestriero. Llm-jepa: Large language models meet joint embedding predictive architectures. *arXiv preprint arXiv:2509.14252*, 2025.

[44] Philip Nicholas Johnson-Laird. *Mental models: Towards a cognitive science of language, inference, and consciousness*. Number 6. Harvard University Press, 1983.

[45] Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

[46] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[47] Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Drrl2gcjzl.

[48] John Langford and Robert Schapire. Tutorial on practical prediction theory for classification. *Journal of machine learning research*, 6(3), 2005.

[49] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023. *URL https://arxiv. org/abs/2211.17192*, 1(2), 2022.

[50] Kenneth Li, Aspen K Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. *ICLR*, 2023.

[51] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. *AI&M*, 1(2):3, 2006.

[52] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024.

[53] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pages 7421–7432, 2024.

[54] Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, Omri Abend, Raz Alon, Tomer Asida, Amir Bergman, Roman Glozman, Michael Gokhman, Avashalom Manevich, Nir Ratner, Noam Rozen, Erez Shwartz, Mor Zusman, and Yoav Shoham. Jamba: A hybrid transformer-mamba language model, 2024. URL https://arxiv.org/abs/2403.19887.

[55] 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.

[56] Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=De4FYqjFueZ.

[57] Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025.

[58] Shuo Liu, Adria Mallol-Ragolta, Emilia Parada-Cabeleiro, Kun Qian, Xin Jing, Alexander Kathan, Bin Hu, and Bjoern W. Schuller. Audio self-supervised learning: A survey, 2022. URL https://arxiv.org/abs/2203.01205.

[59] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

[60] Ashok Vardhan Makkuva, Marco Bondaschi, Adway Girish, Alliot Nagle, Martin Jaggi, Hyeji Kim, and Michael Gastpar. Attention with markov: A curious case of single-layer transformers. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=SqZ0KY4qBD.

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

[62] William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers, 2023. URL https://arxiv.org/abs/2207.00729.

[63] William Merrill, Ashish Sabharwal, and Noah A. Smith. Saturated transformers are constant-depth threshold circuits. *Transactions of the Association for Computational Linguistics*, 10: 843–856, 2022. doi: 10.1162/tacl_a_00493. URL https://aclanthology.org/2022.tacl-1.49/.

[64] William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models, 2025. URL https://arxiv.org/abs/2404.08819.

[65] R Chris Miall and Daniel M Wolpert. Forward models for physiological motor control. *Neural networks*, 9(8):1265–1279, 1996.

[66] Vaishnavh Nagarajan, Chen Henry Wu, Charles Ding, and Aditi Raghunathan. Roll the dice & look before you leap: Going beyond the creative limits of next-token prediction. *arXiv preprint arXiv:2504.15266*, 2025.

[67] Tianwei Ni, Benjamin Eysenbach, Erfan SeyedSalehi, Michel Ma, Clement Gehring, Aditya Mahajan, and Pierre-Luc Bacon. Bridging state and history representations: Understanding self-predictive RL. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ms0VgzSGF2.

[68] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 1525–1534, 2016.

[69] Jongho Park, Jaeseung Park, Zheyang Xiong, Nayoung Lee, Jaewoong Cho, Samet Oymak, Kangwook Lee, and Dimitris Papailiopoulos. Can mamba learn how to learn? a comparative study on in-context learning tasks, 2024. URL https://arxiv.org/abs/2402.04248.

[70] Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. Openwebmath: An open dataset of high-quality mathematical web text, 2023.

[71] Roma Patel and Ellie Pavlick. Mapping language models to grounded conceptual spaces. In *International conference on learning representations*, 2022.

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

[73] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets, 2022. URL https://arxiv.org/abs/2201.02177.

[74] Liliang Ren, Yang Liu, Yadong Lu, Yelong Shen, Chen Liang, and Weizhu Chen. Samba: Simple hybrid state space models for efficient unlimited context language modeling, 2025. URL https://arxiv.org/abs/2406.07522.

[75] Olivier Roy and Martin Vetterli. The effective rank: A measure of effective dimensionality. In *2007 15th European signal processing conference*, pages 606–610. IEEE, 2007.

[76] 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.

[77] Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social iqa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 4463–4473, 2019.

[78] J. Schmidhuber. *Making the World Differentiable: On Using Self Supervised Fully Recurrent Neural Networks for Dynamic Reinforcement Learning and Planning in Non-Stationary Environments*. Forschungsberichte Künstliche Intelligenz. Inst. für Informatik, 1990.

[79] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[80] Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations, 2021. URL https://arxiv.org/abs/2007.05929.

[81] Charlotte Striebel. Sufficient statistics in the optimum control of stochastic systems. *Journal of Mathematical Analysis and Applications*, 12(3):576–592, 1965.

[82] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864.

[83] Jayakumar Subramanian, Amit Sinha, Raihan Seraj, and Aditya Mahajan. Approximate information state for approximate planning and reinforcement learning in partially observed systems. *Journal of Machine Learning Research*, 23(12):1–83, 2022.

[84] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

[85] Yunhao Tang, Zhaohan Daniel Guo, Pierre Harvey Richemond, Bernardo Ávila Pires, Yash Chandak, Rémi Munos, Mark Rowland, Mohammad Gheshlaghi Azar, Charline Le Lan, Clare Lyle, András György, Shantanu Thakoor, Will Dabney, Bilal Piot, Daniele Calandriello, and Michal Valko. Understanding self-predictive learning for reinforcement learning. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[86] Jayden Teoh. Multi-token prediction done right. *arXiv preprint arXiv:2610.38331*, 2026.

[87] Keyon Vafa, Justin Y Chen, Ashesh Rambachan, Jon Kleinberg, and Sendhil Mullainathan. Evaluating the world model implicit in a generative model. *Advances in Neural Information Processing Systems*, 37:26941–26975, 2024.

[88] Keyon Vafa, Peter G Chang, Ashesh Rambachan, and Sendhil Mullainathan. What has a foundation model found? using inductive bias to probe for world models. In *International Conference on Machine Learning*, pages 60727–60747. PMLR, 2025.

[89] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[90] Johannes Welbl, Nelson F Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 94–106, 2017.

[91] Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1819–1862, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.102. URL https://aclanthology.org/2024.naacl-long.102/.

[92] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf.

[93] 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 *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=NRYgUzSPZz.

[94] Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. *Advances in neural information processing systems*, 34:25476–25488, 2021.

[95] 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.

[96] Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020.

[97] Guanhua Zhang, Ricardo Dominguez-Olmedo, and Moritz Hardt. Train-before-test harmonizes language model rankings. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ORv3SAzus1.

[98] Kexin Zhang, Qingsong Wen, Chaoli Zhang, Rongyao Cai, Ming Jin, Yong Liu, James Zhang, Yuxuan Liang, Guansong Pang, Dongjin Song, and Shirui Pan. Self-supervised learning for time series analysis: Taxonomy, progress, and prospects, 2024. URL https://arxiv.org/abs/2306.10125.

A Limitations and Future Work

While NextLat shows strong empirical performance, several limitations remain in our work. First, we do not explore the design space of the latent dynamics model; all experiments use simple MLPs to isolate and demonstrate the effectiveness of the core NextLat approach. More expressive architectures may further improve performance. We also do not study how the width of the hidden layers in the latent dynamics MLP affects learning, even though it effectively acts as a bottleneck that constrains belief-state capacity and may influence performance across tasks. Empirically, we observe that using smaller latent dimensions is beneficial on tasks such as Path-Star graph and Countdown.

Second, the design of the NextLat objective (e.g., stop-gradients, KL self-distillation, Smooth L1 loss) is guided largely by small-scale ablations in Appendix G and empirical intuition. It remains unclear whether multi-step supervision ($d > 1$) and KL token-level supervision are even necessary at larger model and data scales. More systematic studies are needed to better understand how these components interact. Third, due to computational constraints, we did not evaluate against more recent or specialized MTP variants such as the one introduced in Liu et al. [55]. Finally, we did not fully exploit the variable-length nature of NextLat's speculative decoding. In our experiments, the draft length remained fixed throughout decoding for each prompt; we only varied the static draft length between 2 and 10 tokens to identify the configuration with the highest inference speedup. We leave the exploration of more creative adaptive-length speculative decoding strategies for NextLat to future work.

On the analysis side, we do not study the structure of the learned representations under NextLat, leaving open questions about how the method shapes latent spaces. In Section H, we also highlight several quirks observed during pretraining with NextLat, such as increases in Smooth L1 loss ($\mathcal{L}_{\text{next-h}}$) during training and differing loss trajectories across optimizers. These observations suggest that NextLat can be sensitive to optimization dynamics. Better understanding of how to scale and parameterize the NextLat objective remains an important direction for future work.

This work represents only a preliminary study of next-latent prediction, leaving many promising directions for future research. Since the method requires no architectural changes beyond a lightweight latent dynamics model for shaping representations, an interesting direction is to apply it as a post-hoc finetuning objective for pretrained transformers. This could potentially improve reasoning, planning, and world-modeling capabilities of existing models without retraining from scratch. Moreover, because NextLat effectively organizes latent representations with recurrent-like dynamics, an interesting question is whether transformers trained with NextLat are better suited for RL post-training, where value estimation (be it implicit or explicit) benefits from such recursive "Bellman-like" latent structure. Finally, it would be valuable to explore richer latent architectures, such as higher-dimensional or hierarchical belief states spanning multiple layers or tokens, which may further improve long-horizon reasoning and planning.

B Extended Discussions

B.1 Comparison Against Current Approaches

In this section, we discuss the advantages of NextLat compared to prior transformer learning approaches, focusing discussions on belief-learning methods, i.e., BST and JTP. We provide brief descriptions of these methods and their training objectives in Section D. Motivated readers interested in further details beyond the scope of this work may refer to Hu et al. [41] and Ahn et al. [2]. Table 4 summarizes the training and inference parameters, training speed (in iterations per second), and gradient signal characteristics for each method on FineWeb-Edu pretraining, providing context for the discussions that follow.

Computational Costs. While BST benefits from $O(T^2)$ gradient signals per token sequence, this is a double-edged sword. Even with the optimized implementation of Hu et al. [41], training remains extremely costly because gradients must be accumulated over all $O(T^2)$ predictions of different prefix–suffix pairs. Moreover, BST trains two transformer encoders, further increasing compute and parameter cost. During inference, BST also uses both transformer encoders, one for generation and the other for scoring the likelihood of the generated sequence. On FineWeb-Edu pretraining, BST is over $3\times$ slower than NextLat in training speed (see Table 4). Note that this result already uses 10% subsampling of prefix-suffix pairs to prevent out-of-memory issues; the training speed of BST would

be much slower if all pairs were used for training. In contrast, NextLat with $d = 1$ incurs negligible overhead relative to GPT while achieving the same belief-state learning guarantees as BST.

We next compare the compute costs of the multi-step prediction methods, i.e., MTP, JTP, and NextLat. The MTP implementation of Gloeckle et al. [28], as well as other variants such as Liu et al. [55], require additional transformer layers as the prediction horizon d increases, whereas JTP and NextLat keep parameter counts fixed across horizons. JTP and NextLat exhibit similar training speeds for $d = 1$[6], while MTP lags behind. At $d = 8$, JTP is substantially faster than NextLat because NextLat sequentially unrolls its latent dynamics model p_ψ to compute multi-step losses, while JTP computes them in parallel. Nonetheless, this modest increase in computation for NextLat yields substantially better performance than JTP across all benchmarks. Importantly, NextLat's sequential computation remains far more efficient than that of recurrent neural networks (RNNs), which we discuss further in Section B.2.

Data Efficiency and Richer Gradient Signals. A distinctive feature of NextLat is the richness of the learning signal it provides. In previous methods that predict in token space, the learning signal is only anchored to the next token, or multiple tokens. In contrast, NextLat predicts in the latent space. Specifically, NextLat is trained to predict the next hidden state $\mathbf{h}_{t+1}$, which itself parameterizes the entire distribution over X_{t+2}. This shifts supervision from individual token targets to distribution-level alignment. Moreover, because the latent dynamics compose recursively—each latent is predictive of the next— $\mathbf{h}_{t+1}$ implicitly encodes information about $\mathbf{h}_{t+2}, \mathbf{h}_{t+3}, \ldots$. As a result, next-latent supervision not only provides learning signals that are dense in the vocabulary space, but also propagates information about future tokens into earlier representations. Taken together, NextLat replaces sparse token-wise supervision with dense latent supervision, improving data efficiency by extracting more learning signal from each training sequence.

Belief State Learning. GPT and MTP lack any theoretical learning pressure to form belief-state representations, which means that they do not necessarily learn compact representations predictive of future observations. JTP can learn belief states but only under the restrictive condition that the prediction horizon satisfies $d > k$, where k denotes the observability horizon of the underlying data-generating process (see Theorem D.1 in appendix). In long-context sequence modeling, however, the underlying process is often k-observable for a very large k, rendering this condition impractical. NextLat, on the other hand, learns belief-state representations independently of d and larger multi-step prediction horizons are used only to provide richer gradient signals. It also avoids the expensive $O(T^2)$ gradient computation required by BST to learn belief states. As such, NextLat stands as a simple, computationally efficient, and theoretically grounded alternative to existing belief-state learning approaches (i.e., BST and JTP) in autoregressive sequence modeling.

Myopic Nature of Token-level Predictions. Token-level supervision is often myopic. Next-token prediction transformers tend to prioritize short-term dependencies, and studies have shown that early training can often resemble n-gram modeling, which can delay or even prevent learning of the true Markov kernel [17, 60]. Bachmann and Nagarajan [6] further showed that the myopic nature of next-token prediction training can trap models in suboptimal local minima, undermining long-horizon planning. In our experiments, we find that adding additional token-level prediction objectives, as in BST, JTP, and MTP, not only degrades next-token performance but also fails to yield consistent gains in generalization. In contrast, NextLat, which emphasizes latent transition modeling as its primary objective, minimizes degradation in next-token performance and improves downstream generalization by encouraging the learning of structured, predictive representations rather than shallow token-level correlations. In doing so, NextLat offers a compelling alternative to conventional token-level prediction objectives commonly used in transformer training.

B.2 NextLat vs. Recurrent Neural Networks

Algorithmic reasoning requires capabilities most naturally understood through recurrent models of computation, like the Turing machine. However, strict recurrence imposes a sequential computation bottleneck at training time. NextLat introduces a *recurrent inductive bias* for the learned representations via latent transition prediction without turning the transformer into a strictly sequential model.

[6]JTP exhibiting a slightly faster training speed than GPT at $d = 1$ is likely an artifact of `torch.compile()` due to differences in the computation graph.

Computational Efficiency and Relation to Backpropagation Through Time. Traditional RNNs trained using backpropagation through time incur $O(T)$ sequential dependence during training. In contrast, NextLat adds only an additional sequential cost proportional to the rollout horizon $d \ll T$, corresponding to unrolling the latent dynamics model p_ψ for d steps. This allows NextLat to largely retain the transformer's parallel training efficiency while also inheriting a recurrent inductive bias.

Conceptually, the one-step and multi-step prediction in NextLat resembles truncated backpropagation through time (TBPTT) in RNNs, with truncation windows of 1 and d, respectively. However, a key distinction lies in how gradients are propagated. In TBPTT for RNNs, gradient computation is truncated beyond the chosen window, yielding *biased* gradient estimates that lack convergence guarantees. In contrast, the **theoretical convergence of belief-state learning in NextLat, as shown in Theorem 3.2, is irrespective of** d, since it requires only one-step prediction optimality. Intuitively, NextLat performs full backpropagation through the transformer's computation graph, where self-attention connects all past tokens. The latent dynamics model operates in an "outer loop", aligning the transformer's hidden states to be temporally consistent across steps. Crucially, this outer-loop supervision does not truncate gradient flow within the transformer, and therefore avoids bias with respect to d. Larger prediction horizons simply provide richer supervision and faster empirical convergence.

Expressivity of the Recurrence. Modern state-space models (SSMs), such as S4 and Mamba, implement efficient linear recurrent updates while remaining highly parallelizable [31, 30]. In contrast to SSMs, the latent dynamics model p_ψ in NextLat can express nonlinear transitions. Moreover, NextLat does not explicitly perform recurrence in the forward computation. Instead, recurrent-like dynamics emerge implicitly through one-step or multi-step unrolling of the p_ψ and aligning successive hidden states via regression. This induces temporal consistency within the latent space without altering the transformer architecture. However, it is important to note that NextLat modifies the learned representations, not the underlying circuit complexity. The overall computational expressivity of the fixed-depth transformer trained with NextLat remains bounded within the TC^0 complexity class [63]. However, our earlier results in Figure 10 question whether the co-trained latent dynamics model (RNN) is subject to the same TC^0 limitations.

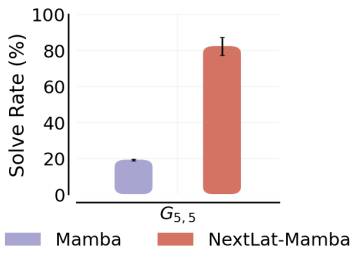


Figure 11: Accuracy on $G_{5,5}$ Path-Star graph task. NextLat improves the performance of Mamba.

Hybrid Attention Models. NextLat differs fundamentally from hybrid Transformer–SSM architectures [69, 54, 74], which explicitly interleave SSM and attention layers. NextLat requires no architectural changes and operates purely as an auxiliary training objective on the model's latent representations. As such, it is broadly compatible with diverse sequence-modeling architectures. In fact, NextLat could complement hybrid models by encouraging compression and consistency in the residual attention pathways that fall outside the SSM's recurrent structure, potentially improving representation efficiency and generalization. NextLat can even be applied to pure SSMs. Although SSMs already compress history through a linear recurrent bottleneck, NextLat adds more explicit pressure toward belief-state learning, along with the data-efficiency benefits mentioned in Section 3.1. In Figure 11, we train Mamba [30] with NextLat on the $G_{5,5}$ Path-Star graph task and find that it significantly improves its solve rate.

C Extended Related Work

We are motivated by a long line of prior works in representation learning for prediction and control. Of close interest are self-supervised learning methods, belief states for decision making, and world models.

Self-Supervised Learning. Self-supervised learning (SSL) is a framework for learning from unlabeled data, where a model generates its own supervisory signals from the structure of raw inputs. Across modalities such as vision, audio, and time series, SSL has proven highly effective for pretraining useful features, enabling downstream transfer that rivals, or even surpasses, models trained on labeled data [58, 7, 98]. There are several approaches to SSL. Our method falls under

self-predictive representation learning, which jointly learns latent representations and a transition function that models how these representations evolve over a sequence. Self-prediction has driven state-of-the-art advances in RL [27, 96, 94, 80, 37]. Herrmann et al. [39] train transformers to predict their own future hidden states through a variational bottleneck, using the resulting prediction error to measure the "interestingness" of in-context computation. However, their predictor operates at an intermediate layer and conditions on the full history of past latents, and therefore does not induce belief states. NextLat instead learns a Markovian, token-conditioned transition over the final hidden states, yielding compact belief-state representations and downstream gains. LLM-JEPA [43] minimizes distances between embeddings of paired text–code data, but relies on manually curated pairs and thus does not generalize to raw text. In contrast, NextLat is fully self-supervised and requires no paired data, making it applicable to transformers across arbitrary sequence modeling domains.

Belief States. In both sequence modeling and RL, models must reason over long histories of observations. To mitigate this curse of dimensionality, prior work focuses on compressing history into latent representations that capture all information necessary for future prediction. In RL, this latent summary is formalized by Kaelbling et al. [46] as a *belief state*, defined as: *"a sufficient statistic for the past history . . . no additional data about its past actions or observations would supply any further information about the current state of the world"*. In stochastic control, the same notion of sufficient statistics appears as "information state" [81]. The idea of sufficient statistics is also key to learning state abstractions in RL [51]. While recurrent neural networks naturally enforce such compression, transformers have no such constraint—their internal state, or memory, grows linearly with sequence length. Recently, Belief State Transformers (BST; [41]) extended the notion of belief states to transformers, and demonstrated benefits in planning tasks. Compared to BST, NextLat learns belief states without requiring a separate transformer and is much more computationally efficient. We compare these methods further in Section 5.

World Models. Loosely, a world model is an internal predictive model of how the world works, with varying interpretations across cognitive science [15, 44], neuroscience [65, 23], control theory [22, 13] and reinforcement learning [84, 78, 33]. Whether transformer language models implicitly learn world models remains a debate; some studies report emergent world understanding [71, 50, 32], while others find incoherent world structure [87, 88]. However, successful world modeling approaches such as *MuZero* [79] for achieving superhuman performance in video and board games, *Dreamer* [34–36] for model-based RL, and *Genie* [10] for interactive video generation share a common principle: they learn a latent dynamics model that takes a latent state (an encoding of past observations), an action, and predicts the next latent state. Yet, learning such latent dynamics for transformer-based language modeling remains underexplored. NextLat addresses this gap by explicitly learning a latent dynamics model that governs how the transformer's latent states evolve given new tokens (i.e., "actions"), enabling the transformer to learn compact latent abstractions of the world with consistent dynamics.

Beyond Next-Token Prediction. In the domain of language modeling, a growing body of work has highlighted the myopic nature of the next-token prediction objective, which limits the model's capability in downstream tasks such as planning and reasoning [6, 66]. Recent works have also found improvements from richer supervision signals that predict further into the future [28, 55, 41, 2]. However, these approaches operate predominantly in the token space. NextLat takes a different approach: it shifts prediction into the latent space, enforcing coherent dynamics over the model's latent representations rather than its token outputs. As we discuss later in Section 3.1, this latent-space supervision provides richer gradient signals than token-level supervision.

Speculative Decoding. Speculative decoding [49, 11] accelerates inference by using a lightweight draft model to propose multiple tokens, which are then verified in parallel by a target model. Closely related, Li et al. [52, 53] train a draft model on top of a frozen language model to predict its high-level latent features. Unlike our approach, this is done post-hoc, whereas we study latent prediction as a pretraining objective to directly shape representations. Moreover, their draft model relies on full attention over past features and therefore does not learn belief states, i.e., compact summaries of past tokens. *Self-speculative decoding* removes the need for a separate draft model by having a single model act as both draft and verifier. A common implementation uses the multi-token prediction (MTP) objective, where the model is trained to predict multiple future tokens that can be verified in

parallel [28]. However, MTP operates in token space and is therefore *typically* constrained to a fixed speculative horizon determined during training. In contrast, NextLat learns a latent dynamics model that can be recursively composed in latent space, enabling *variable-length self-speculative decoding* and faster inference.

D Belief States in Sequence Modeling

Recent work has introduced variants of sequence modeling architectures based on the principle of learning belief states, i.e., BST and JTP. We review these methods here. Let θ denote the parameters of a transformer-based model. Let $\mathbf{h}_{s:t}$ denote the hidden states produced by the transformer encoder for a token sequence $X_{s:t}$, where $s \leq t$. When we use the notation $\mathbf{h}_t$, it is shorthand for $\mathbf{h}_{1:t}$. The model's output head produces a categorical distribution over the token vocabulary conditioned on some hidden state representation, i.e., $p_\theta(\cdot \mid \mathbf{h}_{s:t})$.

Belief State Transformer. The Belief State Transformer (BST; Hu et al. [41]) learns *compact* belief states by jointly training a next-token predictor and a previous-token predictor across all possible prefix–suffix decompositions of a sequence, including cases where either the prefix or suffix is empty. Concretely, given a prefix $X_{1:t}$ and a suffix $X_{t+k:T}$ with $k \geq 1$, BST aims to minimize the cross-entropy loss

$$\mathcal{L}_{\text{BST}}(\theta) = \mathbb{E}_{t<T}\Big[-\log \underbrace{p_\theta(X_{t+1} \mid \mathbf{h}_{1:t}, \mathbf{h}_{t+k:T})}_{\text{next-token prediction}} - \log \underbrace{p_\theta(X_{t+k-1} \mid \mathbf{h}_{1:t}, \mathbf{h}_{t+k:T})}_{\text{previous-token prediction}} \Big], \tag{6}$$

where $\mathbf{h}_{1:t}$ and $\mathbf{h}_{t+k:T}$ are produced by separate transformers. This bidirectional training shapes the hidden representations of the BST into belief states.

Joint Multi-Token Prediction. Joint multi-token prediction (JTP; Ahn et al. [2]) aims to learn the *joint* distribution over the next $d+1$ tokens conditioned on $\mathbf{h}_t$, where d is the multi-step prediction horizon beyond the next token. Specifically, JTP minimizes the loss

$$\mathcal{L}_{\text{JTP}}(\theta; d, \lambda_{\text{MTP}}) = \mathbb{E}_{t<T}\Big[-\log \underbrace{p_\theta(X_{t+1} \mid \mathbf{h}_t)}_{\text{next-token prediction}} - \lambda_{\text{MTP}} \cdot \frac{1}{d} \sum_{i=1}^{d} \log \underbrace{p_\theta(X_{t+i+1} \mid \text{Fetch}(\mathbf{h}_t, X_{t+1:t+i}))}_{\text{joint multi-token prediction}} \Big], \tag{7}$$

where an additional module $\text{Fetch}(\mathbf{h}_t, X_{t+1:t+i})$ is used to create an embedding combining the teacher-forced tokens $X_{t+1:t+i}$ with $\mathbf{h}_t$ and $\lambda_{\text{MTP}} > 0$ balances next- and multi-token prediction losses. Although Ahn et al. [2] suggest that their method learn "short-horizon belief states", they do not formally define the conditions under which this occurs. To understand how JTP learns belief states, we start by defining a k-*observable* system.

Definition D.1 (k-observability for sequences)**.** A system is k-observable if for any two sequences $H = X_{1:t}$ and $H' = X_{1:j}$ that induce the same joint distribution over the next-k tokens, i.e., $\mathbb{P}(X_{t+1:t+k} \mid H) = \mathbb{P}(X_{t+1:t+k} \mid H')$, it follows that their *full-horizon* conditionals match:

$$\mathbb{P}(X_{t+1:T} \mid H) = \mathbb{P}(X_{t+1:T} \mid H'). \tag{8}$$

In other words, the distribution of all future observations is determined by the distribution of the next k observations.

Proposition D.2 (JTP forms belief states in k-observable systems)**.** *Assume the system is k-observable and let $k = d+1$. Suppose the joint multi-token prediction model recovers the true joint next-k conditional, i.e. $p_\theta(X_{t+1} \mid \mathbf{h}_t) p_\theta(X_{t+2} \mid \mathbf{h}_t, X_{t+1}) \dots p_\theta(X_{t+k} \mid \mathbf{h}_t, X_{t+1:t+k-1}) = \mathbb{P}(X_{t+1:t+k} \mid X_{1:t})$ a.s. for all t, then $\mathbf{h}_t$ is a belief state for $X_{1:t}$.*

Proof. By k-observability (Theorem D.1), there exists a measurable map G taking the joint next-k conditional distribution to the full-horizon conditional:

$$\mathbb{P}(X_{t+1:T} \mid X_{1:t}) = G\Big(\mathbb{P}(X_{t+1:t+k} \mid X_{1:t})\Big).$$

By the premise that JTP recovers the true next-k joint, conditioning on $\mathbf{h}_t$ is equivalent to conditioning on $X_{1:t}$ for all bounded measurable functionals of the future. Hence $\mathbf{h}_t$ is a belief state for all $t < T$. □

Intuitively, if all possible futures can be distinguished by the next k tokens, then a JTP model that accurately predicts the joint next-k distribution would encode all information necessary to distinguish future trajectories. Note that both next-token prediction and multi-token prediction do not guarantee belief state representations (see Hu et al. [41]).

E Formal Proof of Theorem 3.2

The proof follows the intuition illustrated below. Optimizing for next-token and transition consistency (Equations 1 and 2) ensures the existence of measurable maps p_θ and p_ψ that allow recursive decoding of future tokens:

$$\mathbf{h}_t \xrightarrow[\text{decode token}]{p_\theta} X_{t+1} \xrightarrow[\text{update state}]{p_\psi} \mathbf{h}_{t+1} \xrightarrow[\text{decode token}]{p_\theta} X_{t+2} \xrightarrow[\text{update state}]{p_\psi} \mathbf{h}_{t+2} \cdots \xrightarrow{p_\theta} X_T.$$

Proof. A formal proof proceeds by backward induction on t. For the base case $t = T-1$, the claim follows directly from Equation 1, since $\mathbf{h}_{T-1}$ suffices to predict X_T.

Now assume $\mathbf{h}_{k+1}$ is a belief state for $X_{1:k+1}$. By Theorem 3.1, this implies that $X_{k+2:T}$ is conditionally independent of $X_{1:k+1}$ given $\mathbf{h}_{k+1}$. We will show that $\mathbf{h}_k$ is also a belief state for $X_{1:k}$. From $\mathbf{h}_k$, one can generate

$$X_{k+1} \sim p_\theta(\cdot \mid \mathbf{h}_k),$$
$$\mathbf{h}_{k+1} \sim p_\psi(\cdot \mid \mathbf{h}_k, X_{k+1}).$$

By next-token and transition consistency (Equations 1 and 2), we have

$$\begin{aligned}
\mathbb{P}(X_{k+1:T} \mid \mathbf{h}_k) &= \mathbb{P}(X_{k+2:T} \mid X_{k+1}, \mathbf{h}_k)\, \underbrace{\mathbb{P}(X_{k+1} \mid \mathbf{h}_k)}_{\text{Equation 1}} \\
&= \mathbb{P}(X_{k+2:T} \mid X_{k+1}, \mathbf{h}_k)\, \mathbb{P}(X_{k+1} \mid X_{1:k}) \\
&= \left[\int \mathbb{P}(X_{k+2:T},\ \mathbf{h}_{k+1} \mid X_{k+1}, \mathbf{h}_k)\, d\mathbf{h}_{k+1}\right] \mathbb{P}(X_{k+1} \mid X_{1:k}) \\
&= \left[\int \underbrace{\mathbb{P}(X_{k+2:T} \mid \mathbf{h}_{k+1}, X_{k+1}, \mathbf{h}_k)}_{\text{Induction hypothesis}} \underbrace{\mathbb{P}(\mathbf{h}_{k+1} \mid X_{k+1}, \mathbf{h}_k)}_{\text{Equation 2}}\, d\mathbf{h}_{k+1}\right] \mathbb{P}(X_{k+1} \mid X_{1:k}) \\
&= \left[\int \mathbb{P}(X_{k+2:T} \mid \mathbf{h}_{k+1})\, \mathbb{P}(\mathbf{h}_{k+1} \mid X_{1:k+1})\, d\mathbf{h}_{k+1}\right] \mathbb{P}(X_{k+1} \mid X_{1:k}) \\
&= \mathbb{P}(X_{k+2:T} \mid X_{1:k+1})\, \mathbb{P}(X_{k+1} \mid X_{1:k}) = \mathbb{P}(X_{k+1:T} \mid X_{1:k}).
\end{aligned}$$

This proves that $\mathbf{h}_k$ is also a belief state. □

F More Details on NextLat Implementation

We parameterize the latent transition model p_ψ with a three-layer MLP using GELU [38] activations. The latent transition model takes as input the layer-normalized [5] concatenation of the current hidden state $\mathbf{h}_t$ and next-token embedding X_{t+1}, and outputs a *delta* update applied via residual connection:

$$\hat{\mathbf{h}}_{t+1} = p_\psi(\mathbf{h}_t, X_{t+1}) = f_\psi(\mathbf{h}_t, X_{t+1}) + \mathbf{h}_t \tag{9}$$

where $f_\psi(\cdot)$ predicts the modification to $\mathbf{h}_t$ (see Figure 12). This paper aims to demonstrate that NextLat yields significant performance gains even with a **simple MLP latent transition**. We foresee even better performances with more sophisticated latent transition model architectures, but we leave that exploration to future work.

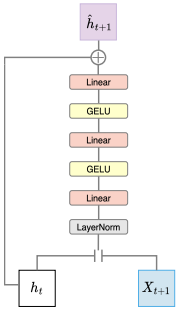


Figure 12: Illustration of the latent transition model p_ψ.

Following standard training, we mask token-level losses (i.e., $\mathcal{L}_{\text{next-token}}$ and $\mathcal{L}_{\text{KL}}$) corresponding to context or prompt tokens. However, we do not apply masking for $\mathcal{L}_{\text{next-h}}$ on context tokens, ensuring that belief state representations develop

even during context processing. When using sequence packing (e.g., for Manhattan, TinyStories, and FineWeb-Edu), we mask $\mathcal{L}_{\text{KL}}$ and $\mathcal{L}_{\text{next-h}}$ terms that cross document boundaries. Because we stop-gradient the output head for the KL objective, we can use the *KL gradient–recovery trick* [86] to yield the KL gradient from a cross-entropy backward pass, without full-vocabulary softmax that explicit KL minimization would require.

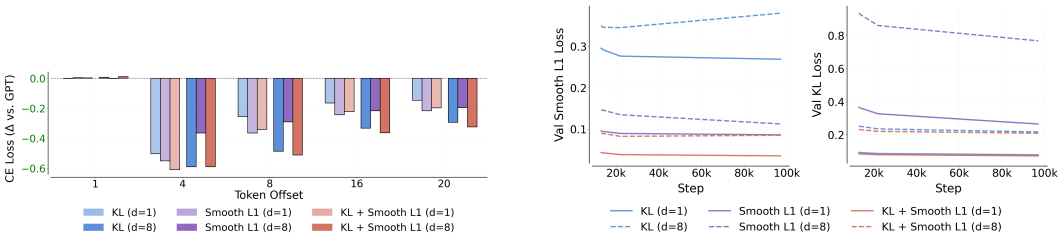
G Ablation Studies

In this section, we ablate the key design choices of NextLat. Specifically, we study the effects of the KL ($\mathcal{L}_{\text{KL}}$) and Smooth L1 ($\mathcal{L}_{\text{next-h}}$) losses, as well as the use of stop-gradients on the target ($\mathbf{sg}[\mathbf{h}_{t+i}]$) in the Smooth L1 loss (Equation 3) and on the output head ($p_\theta^{\mathbf{sg}}(\cdot)$) in the KL loss (Equation 4). We focus our investigations primarily on Countdown and TinyStories, analyzing how these design choices affect the predictive quality of learned hidden states under linear probing and validation loss behavior.

Table 5: Accuracy of $d = 1$ models on Countdown. Best result is **bolded**, and second best is underlined.

Model	Accuracy (%)
GPT	33.1
JTP	39.0
MTP	39.2
BST	42.3
NextLat ($\mathcal{L}_{\text{KL}}$ only)	38.2
NextLat ($\mathcal{L}_{\text{next-h}}$ only)	50.7
NextLat ($\mathcal{L}_{\text{KL}}$ + $\mathcal{L}_{\text{next-h}}$)	**54.8**

We first isolate the effects of the KL and Smooth L1 losses in NextLat on Countdown with $d = 1$. Note that only $\mathcal{L}_{\text{next-h}}$ is required for belief-state convergence, while $\mathcal{L}_{\text{KL}}$ serves as a complementary supervision signal. In Table 5, $\mathcal{L}_{\text{next-h}}$ demonstrates to be the primary driver behind NextLat's gains—using the Smooth L1 loss alone **improves accuracy by 17.6 percentage points over GPT (50.7% vs. 33.1%)** and already outperforms all baselines. In contrast, using the KL loss alone yields only 38.2%. Combining both losses (i.e., the full NextLat design) achieves the highest accuracy of 54.8%, indicating that $\mathcal{L}_{\text{KL}}$ provides a complementary boost on top of $\mathcal{L}_{\text{next-h}}$.



(a) (b)

Figure 13: **KL and Smooth L1 Loss Ablations:** (a) TinyStories probe performance under ablations of the KL and Smooth L1 losses. (b) TinyStories validation Smooth L1 and KL losses.

Next, we ablate the KL and Smooth L1 losses on TinyStories. In Figure 13a, we observe that at $d = 1$, using Smooth L1 loss alone achieves the strongest probe performance 20 tokens ahead relative to GPT on TinyStories. At $d = 8$, however, the combined KL + Smooth L1 objective (i.e., the full NextLat design) performs best, suggesting that the Smooth L1 loss is critical for learning long-range predictive representations, while the KL loss becomes increasingly beneficial at larger horizons. Furthermore, Figure 13b shows that the combined KL + Smooth L1 objective achieves lower validation KL and Smooth L1 losses than optimizing either objective alone. This confirms that the two losses are complementary and provide mutually beneficial supervision.

Next, we compare the KL loss against cross-entropy (CE) as the complementary token-space supervision signal on TinyStories. Note that the CE loss corresponds exactly to the multi-token prediction objective used in JTP and MTP. Unlike the KL loss, which performs self-distillation by matching the predicted token distribution to the model's own softmax distribution under the target hidden state (see Equation 4), the CE loss directly supervises the predicted latent state using the ground-truth next token. As shown in Figure 14a, the original NextLat objective (KL + Smooth L1) achieves substantially better probe performance than CE + Smooth L1 at $d = 1$, though the gap becomes marginal at $d = 8$. Figure 14b shows that across both $d = 1$ and $d = 8$, KL + Smooth L1 achieves

Algorithm 1 Pseudocode for NextLat's loss function in PyTorch syntax. The loss functions `cross_entropy_loss()`, `smooth_l1_loss()`, and `KL_loss()` are implemented externally.

```
import torch.nn as nn
from copy import deepcopy

def loss(batch, targets):
    # batch:(B,T), targets: (B,T)
    # embedding dimension = D, vocabulary size = V
    batch_size, seq_len = batch.shape

    # Encode sequences into latent states
    hidden_states = Transformer(batch) # (B,T,D)

    # Compute next-token loss
    logits_post = Output_Head(hidden_states) # (B,T,V)
    loss_next = cross_entropy_loss(logits_post, targets)

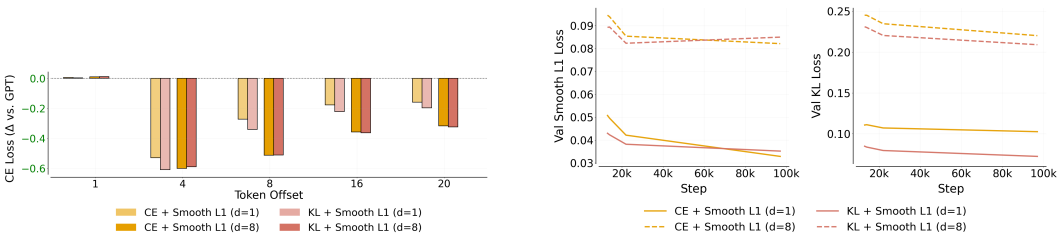
    next_tokens = batch
    next_states = hidden_states
    current_states = hidden_states
    loss_next_h = 0
    loss_kl = 0

    # Recursive multi-step predictions, default multi_step_horizon=1;
    # only 1-step prediction
    for _ in range(multi_step_horizon):
        # Shift hidden states back by 1
        current_states = current_states[:, :-1]
        # Shift next tokens and hidden state targets forward by 1
        next_tokens = next_tokens[:, 1:]
        next_states = next_states[:, 1:]
        logits_post = logits_post[:, 1:]
        # Predict next hidden state using latent dynamics model
        # i.e., (h_t, x_{t+1}) -> \hat{h}_{t+1}
        pred_next_states = Latent_Dynamics(current_states, next_tokens
    ) #(B,T,D)
        # Compute next-hidden loss using detached next states as
    targets
        # i.e., ||h_{t+1} - \hat{h}_{t+1}||^2
        loss_next_h += smooth_l1_loss(pred_next_states, next_states.
    detach())

        # Compute KL loss using detached output head
        # i.e., KL[p(h_{t+1}) || p(\hat{h}_{t+1})]
        logits_prior = deepcopy(Output_Head)(pred_next_states) # (B,T,
    V)
        loss_kl += KL_loss(logits_post.detach(), logits_prior) #
    detach posterior

        current_states = pred_next_states

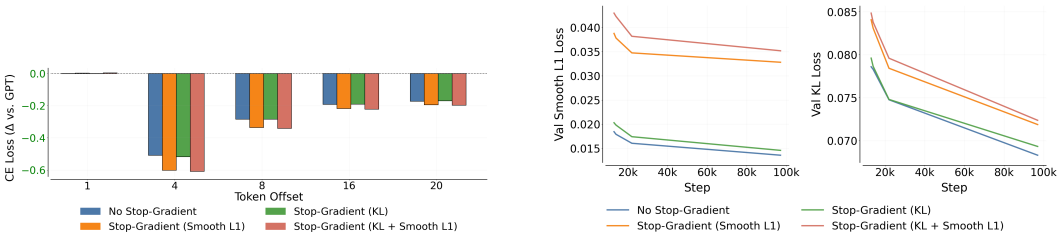
    loss_next_h = loss_next_h / multi_step_horizon
    loss_kl = loss_kl / multi_step_horizon
    # overall NextLat loss
    return loss_next + next_h_lambda * loss_next_h + kl_lambda *
    loss_kl
```



(a) (b)

Figure 14: **KL and CE Loss Ablations:** (a) TinyStories probe performance when replacing the KL loss in NextLat with a CE loss. (b) TinyStories validation Smooth L1 and KL losses.

lower validation KL loss, but slightly worse Smooth L1 loss, than CE + Smooth L1. Overall, these results suggest that at shallow multi-step prediction horizons, the dense distribution-level supervision provided by KL matching induces more predictive representations than CE. However, this advantage appears to diminish at larger horizons. It is also important to note that the KL loss incurs higher memory overhead than CE loss. This is because KL requires materializing full logits and softmax distributions over the vocabulary, whereas modern fused CE implementations only need to materialize probabilities at the target token index. More systematic studies are needed, especially at larger model scales, to determine whether KL or CE is preferable as a complementary supervision signal. For the scope of this work, we stick to using KL loss in all experiments.



(a) (b)

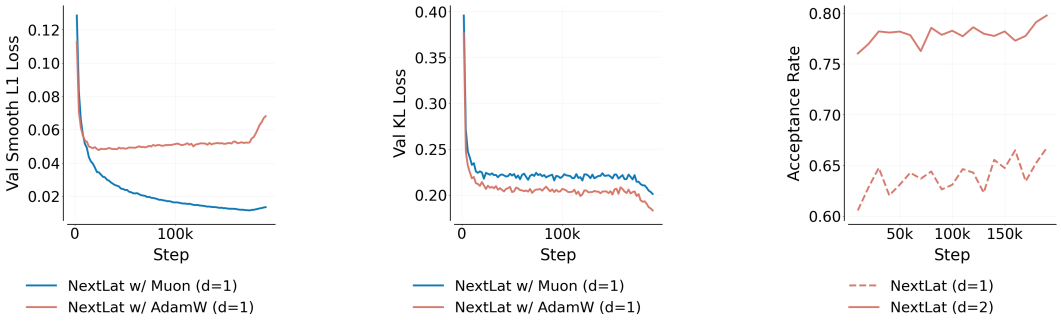
Figure 15: **Stop-Gradient Ablations:** (a) TinyStories probe performance under different stop-gradient configurations. (b) TinyStories validation Smooth L1 and KL losses. Note that all models here use both KL and Smooth L1 losses and differ only in where stop-gradients are applied in those loss components. "Stop-Gradient (Smooth L1)" applies stop-gradient only to the target hidden state in the Smooth L1 loss, "Stop-Gradient (KL)" applies stop-gradient only to the output head in the KL loss, and "Stop-Gradient (KL + Smooth L1)" applies stop-gradients to both components. All models use $d = 1$ here.

Finally, we study the use of stop-gradients in Figure 15 with $d = 1$ on TinyStories. While Figure 15b shows that removing the stop-gradient on the Smooth L1 loss improves validation Smooth L1 and KL losses slightly (~ 0.02), it does not improve the predictive quality of the learned representations. As shown in Figure 15a, applying stop-gradients to both the target in the Smooth L1 loss and the output head ($p_\theta^{\text{sg}}(\cdot)$) in the KL loss yields the best probing performance across all token offsets. Stop-gradients also improve efficiency by reducing the number of backward passes through the transformer. In particular, the KL loss introduces an additional forward and backward pass through the output head, which can be computationally expensive in language models due to large vocabulary sizes[7]. Removing this extra backward pass significantly improves both speed and memory efficiency. Empirically, on tasks like Countdown and Path-Star graph, we also observe that the stop-gradient (especially on the Smooth L1 loss) is essential for high accuracy.

Overall, these ablations suggest that NextLat's effectiveness arises from the combination of Smooth L1 and KL losses together with carefully placed stop-gradients, all of which contribute to learning predictive belief-state representations in transformers.

[7]Note that multi-token prediction models also incur an additional forward+backward pass through the output head for each extra token prediction.

H NextLat Pretraining Quirks



(a) (b) (c)

Figure 16: **FineWeb-Edu Pretraining Curves:** (a) Validation smooth L1 loss during training. (b) Validation KL loss during training. (c) Speculative decoding acceptance rate over a 5-token lookahead horizon (excluding the next-token prediction).

In this section, we expose some optimization quirks that we faced during pretraining with NextLat. First, Figure 16a shows vastly different Smooth L1 loss trajectories when using Muon [45] versus AdamW [59] optimizer under the same learning rate schedule (using the Muon update scaling rule proposed by Liu et al. [57]). One possible explanation is that Muon's orthogonalized update rule enables more efficient latent representation shaping and next-latent prediction optimization. However, Figure 16b shows that AdamW achieves better KL loss optimization than Muon.

Importantly, both optimizers exhibit **an increase in Smooth L1 loss near the end of training** corresponding to the learning rate cooldown stage, though the effect is substantially more pronounced for AdamW. The cause of this behavior remains unclear. We explored several potential fixes to no avail, including:

1. replacing Smooth L1 with MSE loss,
2. removing stop-gradients from $\mathcal{L}_{\text{next-h}}$,
3. tuning loss coefficients $\lambda_{\text{next-h}}$ and λ_{KL},
4. disabling learning rate decay for the latent transition model, and
5. disabling weight decay for the latent transition model.

We leave a deeper investigation of this phenomenon to future work. To our reassurance, the increase in smooth L1 loss does not appear to degrade the coherence of the latent transition model. As shown in Figure 16c, the speculative decoding acceptance rate over a lookahead horizon of 5 tokens continues to improve throughout training, even during the phase where the smooth L1 loss increases under AdamW. This suggests that the rise in smooth L1 loss may reflect changes in the scale or geometry of the latent states, rather than degradation in the token-level coherence of the latent transition dynamics.

On a side note, we observed gradient norm spikes that led to training instability when training MTP with Muon, which unfortunately limited our pretraining analysis to AdamW (despite Muon showing a more favorable learning trajectory with NextLat).

I Extra Experiment Details

In this section, we provide additional experimental setup details and supplementary results for the experiments in the main body. All experiments were run on NVIDIA RTX A5000, NVIDIA H100 NVL, and NVIDIA B200 GPUs. Table 6 summarizes the training, model, and NextLat hyperparameters across all experimental domains.

The hyperparameters reported in Table 6 were chosen through a small-scale search, exploring $\lambda_{\text{next-h}} \in \{1.0, 2.0\}$ and $\lambda_{\text{KL}} \in \{0.1, 1.0\}$, guided by empirical observations and intuition. Encouragingly, we find that NextLat performs robustly over a wide range of settings. In particular, λ_{KL} requires minimal tuning; $\lambda_{\text{KL}} = 1.0$ works decently across all tasks. After all, it primarily serves as a complementary alignment objective. Likewise, $\lambda_{\text{next-h}} = 1.0$ is effective in most cases, though slightly higher values

Table 6: Training, Model, and NextLat hyperparameters across all benchmarks.

	Manhattan	Countdown	Path-Star	TinyStories	A_5	FineWeb-Edu
Training Parameters						
Steps	400k (6 epochs)	100k (204 epochs)	20k (51 epochs)	100k (8.5 epochs)	400k (512 epochs)	191k (100B tokens)
Batch Size	256	1024	512	256	1024	512
Learning Rate	1e-4	3e-4	5e-4	3e-4	1e-4	4e-4
Learning Rate Schedule	Constant	Constant	Constant	Constant	Constant	WSD
Weight Decay	0.01	0.1	0.1	0.1	0.01	0.1
Clip Gradient Norm	-	1	100	1	1	1
Optimizer	*AdamW* [59] with $\beta_1 = 0.9,\ \beta_2 = 0.95$					
Model Parameters						
Layers	48	12	12	8	2	22
Heads	8	12	6	8	8	16
Hidden Dimension	384	768	384	768	512	2048
NextLat Parameters						
$\lambda_{\text{next-h}}$	1.0	2.0	1.0	1.0	1.0	1.0
λ_{KL}	0.1	1.0	1.0	1.0	0.0	1.0
p_ψ MLP Hidden Dimension	1536	768	384	2048	1024	6528
p_ψ MLP Layers	3	3	3	3	3	3

(e.g., $\lambda_{\text{next-h}} = 2.0$) was beneficial when the next-latent regression loss is of much smaller scale than the token-level losses (i.e., $\mathcal{L}_{\text{next-token}}$ and $\mathcal{L}_{\text{KL}}$).

For MTP and JTP, we sweep multi-token prediction loss weights $\lambda_{\text{MTP}} \in \{0.1, 0.2, 0.4, 0.6, 0.8, 1.0\}$ to ensure fair baseline comparisons. For FineWeb-Edu pretraining, extensive hyperparameter tuning is computationally prohibitive, so we use uniform loss weights across all methods (i.e., $\lambda_{\text{next-h}} = \lambda_{\text{KL}} = \lambda_{\text{MTP}} = 1.0$) to provide the fairest comparison possible under our compute constraints.

I.1 Manhattan Taxi Rides

Here, we provide additional details on our training and evaluation setups for the Manhattan Taxi Rides benchmark and clarify key differences from the original study of Vafa et al. [87].

Training. Since this task inherently requires state tracking (i.e., tracking position within Manhattan), and increasing model depth is known to benefit transformers on such tasks [64], we employ 48-layer transformers with 384 hidden dimensions and 8 attention heads (88M parameters). This differs from Vafa et al. [87], which used 12 layers, 768 hidden dimensions, and 12 heads for their smaller transformer variant. We found that increasing model depth yielded substantial performance gains, whereas increasing hidden dimensionality offered negligible improvement. As shown in Table 1, the effective latent rank of our models is substantially smaller than 384, suggesting that large hidden dimensions are unnecessary.

Unlike the original study, which trained models for only one epoch, our models are trained for six epochs, as we observed that performances generally do not converge within a single epoch. This also helps rule out potential "grokking" phenomena [73], where generalization improves only after extended periods of overfitting. To enable longer training without substantially increasing runtime, we apply sequence packing, i.e., concatenating multiple sequences into longer ones while masking cross-sequence attention. This enables efficient utilization of GPU memory and computation. Most models complete six training epochs in under three days on a single NVIDIA H100 NVL GPU, except MTP, which has substantially more parameters than the others.

Evaluation. Models are trained on random traversals of length 100 connected pickup and dropoff intersections. This task inherently requires not only state tracking but also planning, as models must reason over possible future paths to generate valid 100-step trajectories that reach the destination while avoiding dead ends, i.e., road segments disconnected from the goal due to the one-way streets. Random traversals rarely correspond to true shortest paths and do not provide an inductive bias toward learning the shortest path algorithm, which relies on dynamic programming. Consequently, unlike Vafa et al. [87], who evaluated pairs with shortest paths of up to 100 steps, we limit evaluation pairs to paths of up to 50 steps. This adjustment ensures that evaluation pairs do not demand long-horizon planning beyond the training distribution, which might otherwise force the model to produce forced predictions to compensate for planning failure. This also ensures that inconsistencies in a model's internal map are more reflective of world-model incoherence rather than artifacts of long-horizon planning limitations. These evaluation pairs are used to generate Figure 3 and to compute the sequence compression and detour robustness metrics, following the procedure of Vafa et al. [87].

For the effective latent rank, we pass a batch of 256 sequences (each of length 256) through the model to obtain the hidden state matrix. Singular values smaller than $1e{-}12$ are discarded, and the effective rank is then computed following Roy and Vetterli [75]. For GPT and NextLat, we use the final-layer hidden states. For JTP, we extract the hidden states immediately before the self-attention module in the Fetch head (see Equations 4–5 in Ahn et al. [2]). For MTP, we use the output of the next-token prediction head to compute the effective rank.

I.2 Countdown

We largely follow Gandhi et al. [25] for the Countdown training and evaluation setup. Each problem consists of four input numbers and a solution sequence comprising three equations, consistent with prior work [25, 93]. A training example is formatted as

$$\underbrace{14, 83, 88, 91,}_{\text{inputs}} \overbrace{23}^{\text{target}} \mid \underbrace{83 - 14 = 69,\ 91 - 88 = 3,\ 69/3 = 23}_{\text{solution}}$$

where the first four numbers are the inputs, the fifth is the target number, and the pipe symbol "|" separates the input prompt from the solution. During training, loss values corresponding to input prompt are masked out.

Previous studies involving the Countdown benchmark used pretrained GPT-2 byte-pair encoding tokenizers, which do not necessarily tokenize multi-digit numbers as single units. In contrast, we construct a custom tokenizer that assigns each integer from 1 to 10,000 to a unique token, ensuring that every number in the sequence is represented atomically. The arithmetic operators and delimiters, i.e., $\{\ |\ ,\ +\ ,\ -\ ,\ \times\ ,\ \div\ \}$, are each assigned their own token indices. Due to the large branching factor of the Countdown problem, we insert eight pipe symbols ("|") between the input and the solution as pause tokens [29], allowing the model additional computation steps to plan before generating its answer.

I.3 Path-Star

Our Path-Star data preparation, training, and evaluation follow Bachmann and Nagarajan [6], except that we increase the weight decay to 0.1, which we found helpful for stable convergence and higher solve rates in the multi-step prediction methods (i.e., MTP, JTP, and NextLat). We evaluate each model's ability to generate the correct arm on 20k held-out test instances. Unlike Hu et al. [41] and Ahn et al. [2] which generate a fresh set of graphs every batch, we adopt the original, more challenging setup of Bachmann and Nagarajan [6], which uses a fixed sample size of 200k and node values sampled from $N = 100$. This difference accounts for the performance gap observed in the BST and JTP baselines in Figure 6. The Path-Star experiment is designed to expose the myopic behavior of teacher-forced next-token prediction, which can encourage models to exploit superficial regularities—an effect referred to as the *Clever Hans cheat* [6]. Because the task's sample space grows exponentially with graph size, identifying the correct algorithm that generalizes across all graph instances is highly nontrivial. While not conclusive, our results suggest that latent-space prediction and the inductive bias toward compressing history into belief states promote better discovery of generalizable solutions in data-constrained settings.

I.4 TinyStories

Our TinyStories setup follows exactly Hu et al. [41]. After pretraining on TinyStories, we train linear (one-layer) probes on the hidden states of the frozen transformer models for an additional 20k steps on the same dataset. All probe training hyperparameters (e.g., learning rate, batch size) match those used during pretraining (see Table 6).

For GPT, JTP, and NextLat, the choice of hidden states follows the setup used for measuring effective rank in the Manhattan taxi rides task, as described in Section I.1. For BST, we use the final-layer hidden states of the forward transformer encoder. For MTP [28], we use the output of the shared transformer trunk, i.e., the hidden state before it branches into separate transformer heads for multi-token prediction, as this final shared representation contains the most predictive information about future tokens.

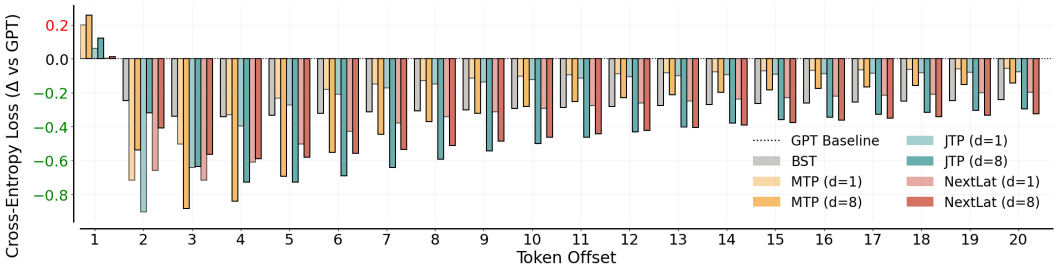


Figure 17: Full plot version of Figure 8 that shows probe performance across all 20 tokens offsets.

I.5 A_5 State Tracking

The group A_5 consists of the even permutations of 5 elements. Intuitively, each token in the sequence represents a permutation operation that rearranges the five elements, and after each operation the model must output the resulting arrangement (i.e., state). The task is therefore fundamentally a *state-tracking* problem: the model must maintain and update an internal representation of the current state as new permutation operations are sequentially composed.

Our training setup follows Merrill et al. [64]. We trained the models on 1 million unique 12-token sequences over the group A_5. We then evaluated their length generalization capabilities on approximately 100k 36-token sequences. For NextLat training, we used **one-step supervision** ($d = 1$) and optimized only the regression objective ($\lambda_{\text{next-h}} = 1$), without token-level supervision ($\lambda_{\text{KL}} = 0$). This ensures that any observed length generalization of the latent dynamics model (RNN) arises purely from faithful next-latent prediction rather than auxiliary multi-token prediction signals.

We also experimented with transformers without positional embeddings (NoPE), motivated by prior work suggesting improved length generalization [47]. However, we found that NoPE degraded performance in this setting, so all experiments use Rotary Position Embeddings (RoPE; Su et al. [82]). Unlike RNNs, whose hidden states evolve sequentially and therefore implicitly encode token order, transformers process sequences in parallel and rely heavily on positional embeddings to represent token order information during training. Interestingly, despite positional embeddings being explicitly included in the transformer hidden states, the latent dynamics model trained on one-step latent transitions derived from the transformer still generalizes beyond the training sequence length. This suggests that the learned transition dynamics is robust to the transformer's positional encodings.

I.6 FineWeb-Edu Pretraining

Following standard practices, all models are optimized with AdamW using a peak learning rate of $4e{-}4$, weight decay of 0.1, and gradient clipping at 1.0. We use a global batch size of 0.5M tokens and a sequence length of 1024 tokens. The learning rate follows a Warmup-Stable-Decay (WSD) schedule [42], consisting of a 1B-token linear warmup followed by a 10B-token linear decay phase. All models use the GPT-2 tokenizer with a vocabulary size of 50,257.

We evaluated the zero-shot accuracy of the pretrained models on nine standard language modeling benchmarks: Wikitext (Wiki; Merity et al. [61]), LAMBADA (LAMB.; Paperno et al. [68]) (standard version), PIQA [8], HellaSwag (HellaS.; Zellers et al. [95]), WinoGrande (Wino.; Sakaguchi et al. [76]), ARC-easy (ARC-e) and ARC-challenge (ARC-c) [12], Social IQa (SIQA; Sap et al. [77]), and SciQ [90]. We also evaluate the self-speculative decoding performance of JTP, MTP, and NextLat across Wikipedia [21], Books (BookCorpusOpen; Face [19]), Code (Stack-Edu; Allal et al. [3]), and Math (OpenWebMath; Paster et al. [70]) domains. To do so, we sample 1024 prompts of length 512 tokens from each dataset and generate 512-token continuations using the speculative sampling algorithm of Leviathan et al. [49]. We then measured the speedup in inference of each model using self-speculative decoding compared to naive autoregressive sampling from the transformer, measured on $8\times$NVIDIA B200 GPUs.

I.6.1 Results with MTP/JTP (d=4)

Table 7: Downstream language modeling evaluation on 1.3B-parameter models trained on 100B FineWeb-Edu tokens. Best scores are in bold and second-best are underlined.

Model	FW-Edu ppl ↓	Wiki. ppl ↓	LAMB. ppl ↓	LAMB. acc ↑	PIQA acc ↑	HellaS. acc ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc ↑	SIQA acc ↑	SciQ acc ↑	Avg.
GPT	**10.52**	**17.93**	20.26	42.07	73.45	**58.79**	60.46	68.18	39.16	42.32	86.10	58.82
JTP (d=1)	11.08	19.28	21.88	41.35	**74.92**	57.43	58.64	68.73	39.25	42.99	87.30	58.83
JTP (d=2)	11.18	19.60	22.11	41.37	73.34	56.84	59.98	68.86	38.57	**43.35**	86.70	58.63
JTP (d=4)	11.29	20.45	20.65	41.90	73.45	56.58	57.70	69.73	39.68	42.53	**88.50**	58.76
MTP (d=1)	10.90	18.82	20.23	41.26	74.32	58.05	**60.54**	68.52	38.91	42.84	85.40	58.73
MTP (d=2)	11.00	18.61	18.34	43.43	72.80	57.92	59.35	68.35	39.08	41.97	86.60	58.69
MTP (d=4)	11.10	18.97	22.75	40.69	73.72	57.39	58.33	**70.20**	39.51	42.12	85.90	58.48
NextLat (d=1)	10.83	18.39	19.77	41.08	73.07	58.35	59.27	69.65	39.68	43.24	86.00	58.79
NextLat (d=2)	10.88	18.44	**17.83**	**43.86**	73.61	57.79	59.20	69.74	**40.10**	41.91	87.50	**59.21**

In this section, we extend our comparisons to include JTP and MTP trained with larger horizons ($d = 4$). Note that the training cost of JTP and MTP increases substantially with larger d (see Table 4), making JTP ($d = 4$) and MTP ($d = 4$) significantly more expensive to train than NextLat ($d = 1, 2$).

Table 7 shows that increasing the multi-token prediction horizon to $d = 4$ still does not yield meaningful improvements on multiple-choice benchmarks; NextLat ($d = 2$) continues to achieve the best average accuracy overall. Table 8 shows that even when JTP and MTP are trained to be able to draft more tokens (i.e., 4 tokens ahead), they still fail to surpass the variable-length speculative decoding performance of NextLat across most domains, with the exception of the Code domain. Finally, Figure 18 shows that, on the FineWeb-Edu validation set, increasing the training horizon to $d = 4$ yields only modest speedup gains for JTP and MTP. Their speedup curves saturate earlier and remain below that of NextLat ($d = 2$), which continues to improve with longer draft lengths and achieves the highest overall speedup.

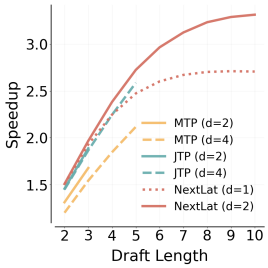


Figure 18: Inference speedup ratio on the FineWeb-Edu validation set.

Table 8: Relative speedup and average accepted tokens per drafting step over diverse domains. Note that "Accepted Tokens" excludes the next-token prediction which is always accepted.

	Wikipedia		Books		Code		Math	
Model	Speedup	Accepted Tokens	Speedup	Accepted Tokens	Speedup	Accepted Tokens	Speedup	Accepted Tokens
JTP (d=1)	1.46	0.96	1.47	0.97	1.47	0.98	1.46	0.97
JTP (d=2)	1.88	1.84	1.90	1.89	1.88	1.85	1.89	1.86
JTP (d=4)	2.58	3.32	2.62	3.43	**2.61**	3.43	2.42	3.02
MTP (d=1)	1.38	0.91	1.39	0.95	1.40	0.97	1.39	0.95
MTP (d=2)	1.68	1.72	1.72	1.83	1.75	1.91	1.72	1.84
MTP (d=4)	2.10	3.04	2.25	3.44	2.32	**3.68**	2.25	3.46
NextLat (d=1)	2.68	3.52	2.72	3.64	2.29	2.66	2.30	2.72
NextLat (d=2)	**3.21**	**4.59**	**3.32**	**4.86**	2.38	2.83	**2.87**	**3.94**

NeurIPS Paper Checklist

1. **Claims**
   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?
   Answer: [Yes]
   Justification: We've described our claims and contributions clearly in the abstract and introduction.
   Guidelines:
   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.
2. **Limitations**
   Question: Does the paper discuss the limitations of the work performed by the authors?
   Answer: [Yes]
   Justification: We included the limitations and future directions of our paper under Section A of the appendix.
   Guidelines:
   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.
3. **Theory assumptions and proofs**
   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?
   Answer: [Yes]
   Justification: Yes, each theoretical result is stated with its assumptions, and complete proofs are provided (typically in the appendix).
   Guidelines:
   - The answer [N/A] means that the paper does not include theoretical results.

- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provided all information, i.e. hyperparameters, libraries, and algorithms, required to reproduce our results in Section I

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Our code is available at `https://github.com/JaydenTeoh/NextLat`.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not

including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**
   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?
   Answer: [Yes]
   Justification: We provided full implementation details in the main body and appendix.
   Guidelines:
   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.
7. **Experiment statistical significance**
   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?
   Answer: [Yes]
   Justification: When possible (i.e., excluding pretraining experiments), our results are obtained across multiple independent runs, and we plotted the results with variance.
   Guidelines:
   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.
8. **Experiments compute resources**
   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?
   Answer: [Yes]
   Justification: We provided the hardware details (GPUs) in the appendix.
   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**
   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?
   Answer: [Yes]
   Justification: We've read the code of ethics.
   Guidelines:
   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).
10. **Broader impacts**
    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?
    Answer: [N/A]
    Justification: Our paper proposes a new way to train transformer models. Societal impacts are limited and thus we omitted this discussion in our paper.
    Guidelines:
    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).
11. **Safeguards**
    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?
    Answer: [N/A]
    Justification: Our paper does not have such risks as we will not be releasing any pre-trained models or new datasets.
    Guidelines:
    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring

that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**
Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?
Answer: [Yes]
Justification: All existing assets are publicly released and used in accordance with their respective licenses; we cite the original sources in the paper.
Guidelines:
    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.
13. **New assets**
Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?
Answer: [N/A]
Justification: We don't release any new assets.
Guidelines:
    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.
14. **Crowdsourcing and research with human subjects**
Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?
Answer: [N/A]
Justification: Our research does not involve human subjects.
Guidelines:
    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.
15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**
Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals

(or an equivalent approval/review based on the requirements of your country or institution) were obtained?
Answer: [N/A].
Justification: Our paper does not involve crowdsourcing nor research with human subjects.
Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**
Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.
Answer: [N/A]
Justification: LLM is only used for writing.
Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.