DLL: DEEP LATENT LEARNING VIA LAYER-WISE LATENT SUPERVISION

Anonymous authors
Paper under double-blind review

ABSTRACT

Traditional autoregressive predictive learning paradigms rely primarily on surface-level observable objectives, such as next discrete tokens or future continuous values. This paper argues that observable samples induce additional targets in the deep latent space, namely the contextualized representations embedded in deep layers. We propose **Deep Latent Learning (DLL)**. DLL trains the model to simultaneously predict what the next observable target is and how that target should be represented in the deep latent space. We emphasize the necessity of supervising deep layer-wise representations, as deep layers leverage sequence-wide context through attention mechanisms far more effectively than shallow layers. To complement deep latent supervision, DLL introduces a self-purification module that anchors latent targets to the observables to prevent representation collapse. DLL yields robust improvements on established benchmarks across three domains, such as ImageNet-1K in computer vision, TableBench in natural language processing, and GIFT-Eval in time series forecasting.

1 INTRODUCTION

Classical supervised learning is bound to labeled pairs (x, y), where y is a target with explicit guidance defined by humans or specific tasks. In autoregressive tasks, self-supervised learning expands the sources of supervised objectives: instead of relying on external labels, it explores training targets from x itself through masked input, next-token prediction, or next-step prediction (Devlin et al., 2019; Radford et al., 2019; Brown et al., 2020; He et al., 2022). However, mainstream self-supervised objectives still primarily come from surface-level observable targets. In computer vision, targets are typically future pixel values; in language models, they are future discrete tokens; in time series forecasting, they are future numerical values or data patches. While surface-level objectives have proven fruitful in autoregressive self-supervised tasks, they inherently suffer from surface noise and task-irrelevant stochasticity. Raw surface observations—such as pixel-level illumination fluctuations in vision, sub-word lexical variations in text, or sensor measurement noise in time series—are impaired by noisy details that do not reflect the underlying state dynamics. In contrast, latent space supervision focuses on continuous structural transitions rather than surface-level fluctuations in autoregressive tasks (Baevski et al., 2022; Assran et al., 2023).

We introduce **Deep Latent Learning (DLL)**, which harmonizes supervision from both observable and deep latent prediction targets. This dual-objective design allows the model to capture high-level contextual semantics without sacrificing fine-grained observable details.

However, a fundamental architectural question remains: where within the network hierarchy should latent targets be extracted? We argue the necessity of supervising layer-wise representations, as deeper layers capture increasingly richer contextual semantics while the underlying representation space continuously evolves across layers. Mirroring the historic development from static word embeddings (e.g., Word2Vec (Mikolov et al., 2013)) to fully contextualized representations (e.g., BERT (Devlin et al., 2019)), we highlight the critical role of contextual incorporation within deeper model layers (Ethayarajh, 2019).

Whereas deep latent targets provide richer supervisory signals, it is essential that they remain anchored to observable targets to prevent drifting into degenerate states. This anchoring is particularly critical for deep-layer latent space, as representations naturally diverge from raw surface targets

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

with increasing network depth (Grill et al., 2020; Jing et al., 2022), making them increasingly susceptible to representation collapse. Our approach introduces a lightweight self-purification module combined with the stop-gradient mechanism. The self-purification module *purifies* the latent targets by anchoring them to observable targets so that high-level representations do not become unconstrained. The stop-gradient mechanism severs collapsing feedback loops during backpropagation (Chen & He, 2021).

Through extensive experiments, we demonstrate that DLL achieves robust improvements across three domains:

- **Computer Vision:** DLL outperforms other top-tier ViT-B/ViT-L backbone models like MAE (He et al., 2022), JEPA (Assran et al., 2023) and DINO (Caron et al., 2021), on the ImageNet-1K benchmark (Russakovsky et al., 2015);
- **Natural Language Processing:** DLL achieves a higher score than baseline models on the TableBench benchmark (Wu et al., 2025) and cybersecurity benchmarks CTI-MCQ and CTI-RCM (Alam et al., 2024);
- **Time Series Forecasting:** DLL surpasses the baseline version during post-training with Toto-2.0-313m and Toto-2.0-2.5B (Khwaja et al., 2026) on the GIFT-Eval benchmark (Aksu et al., 2024), and achieves superior forecasting performance in experiments on Chronos datasets (Ansari et al., 2024) and TSlib (Wu et al., 2023).

2 Related Work

Latent Representation Prediction. Joint-embedding predictive architectures motivate prediction in representation space rather than reconstructing surface-level observations. Image Joint-Embedding Predictive Architecture (I-JEPA) and Next-Embedding Prediction Autoregression (NEPA) validate this idea in image representation learning (Assran et al., 2023; Xu et al., 2025; LeCun, 2022). While I-JEPA solely leverages the final layer output as its supervisory objective, DLL enforces layer-wise supervision, directly benefiting from the progressive representation evolution across network layers (Assran et al., 2023; Ethayarajh, 2019). Furthermore, I-JEPA remains a masked non-autoregressive paradigm like Masked Autoencoders (MAE), while DLL adopts a next-step autoregressive architecture (Assran et al., 2023; He et al., 2022). Whereas NEPA also adopts a next embedding prediction paradigm (Xu et al., 2025), it employs solely the first-layer embedding as the training objective. Meanwhile, DLL enforces layer-wise latent supervision that encompasses deeper network layers, effectively capturing the rich, sequence-wide contextual semantics that progressively emerge across depth (Xu et al., 2025; Ethayarajh, 2019).

Representation Collapse and Target Stabilization. A fundamental challenge in latent space learning is representation collapse, where the network output degenerates into trivial states. In order to prevent representation collapse, I-JEPA (Assran et al., 2023) and BYOL (Grill et al., 2020) employ an asymmetric architecture between the input and target encoders via Exponential Moving Average (EMA (Tarvainen & Valpola, 2017)) on the target encoder. In contrast, DLL introduces a self-purification module that keeps the latent targets anchored to the observables, intrinsically preventing representation collapse. Consequently, DLL eliminates the substantial memory overhead of maintaining an EMA target network. Whereas NEPA (Xu et al., 2025) primarily leverages the stop-gradient mechanism on the target embedding to prevent degenerate trivial solutions, DLL also employs a stop-gradient on the latent targets. Alternative approaches, such as VICReg (Bardes et al., 2022), enforce explicit variance-covariance regularizations on embeddings.

Deep Feature Dynamics and Contextual Aggregation. The spatial evolution of internal representations across Transformer layers has been extensively studied. It is revealed that Transformer architectures naturally mirror classical processing pipelines, where shallow layers encode local, surface-level syntax while higher layers progressively consolidate complex, sequence-wide contextual semantics (Tenney et al., 2019). It is also demonstrated that representation contextualization increases with network depth, alongside an inherent phenomenon where deeper-layer representations tend to cluster along a few dominant directions in the latent space (i.e., anisotropy) (Ethayarajh, 2019). These insights collectively motivate the design of DLL: supervising shallow layers alone remains limited in capturing high-level contextual transitions, whereas enforcing supervision

directly over deep layer-wise targets explicitly capitalizes on high-level semantic aggregation across network layers.

3 METHOD

Figure 1 illustrates the training framework of **Deep Latent Learning (DLL)**. DLL augments a standard next-step autoregressive predictor with deep latent target supervision during training. The model is trained to predict both the next observable target and the next layer-wise latent representations. The self-purification module keeps the next layer-wise latent representations anchored to the next observable target. DLL-specific modules are used only during training and introduce no inference-time cost.

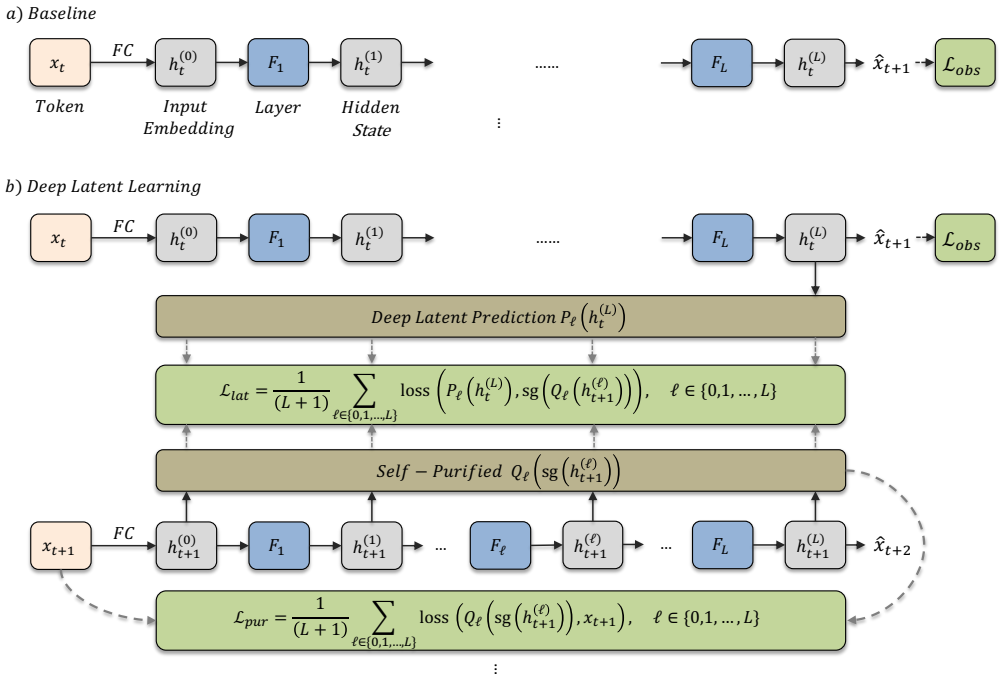


Figure 1: **DLL training framework**. *a) Baseline* predicts the next observable target from the inputs of the current step and previous time steps, supervised by the observable loss $\mathcal{L}_{\text{obs}}$. *b) Deep Latent Learning* additionally feeds the next-step ground-truth target through the layers to obtain layer-wise target representations. The self-purification module $Q_\ell(\cdot)$ anchors these latent targets to the observable targets with the self-purification loss $\mathcal{L}_{\text{pur}}$. The deep latent prediction module $P_\ell(\cdot)$ is applied on the last-layer hidden state of the current step to predict the self-purified latent targets of the next step, supervised by the latent loss $\mathcal{L}_{\text{lat}}$. For simplicity, $Q_\ell(\cdot)$ and $P_\ell(\cdot)$ can be a linear layer or an MLP.

3.1 OBSERVABLE TARGETS PREDICTION

The baseline model uses the output $\hat{x}_{t+1}$ at the current step t as prediction of the observable target x_{t+1} at the next step $t+1$, supervised with the observable loss $\mathcal{L}_{\text{obs}}$:

$$\mathcal{L}_{\text{obs}} = \text{loss}(\hat{x}_{t+1}, x_{t+1})\,. \tag{1}$$

The indices t and $t+1$ are used only for exposition; the model operates on the full sequence.

3.2 SELF-PURIFICATION MODULE ON LATENT TARGETS

While DLL preserves the observable loss $\mathcal{L}_{\text{obs}}$, it extends supervisory objectives into deep layer-wise latent representations. Suppose we have a causal decoder backbone with layers $\ell \in \{1, \dots, L\}$. Let

$E(\cdot)$ be the input embedding or the output of layer 0, $F_\ell(\cdot)$ be the function of layer ℓ, and $h_{t+1}^{(\ell)}$ be the hidden state as the output of layer ℓ at the next step $t+1$:

$$h_{t+1}^{(0)} = E(x_{t+1}), \qquad h_{t+1}^{(\ell)} = F_\ell\Big(h_{t+1}^{(\ell-1)}\Big), \quad \ell \in \{1, \dots, L\}. \tag{2}$$

Here, $F_\ell(h_{t+1}^{(\ell-1)})$ stands for the step-$(t+1)$ output of layer ℓ under the architecture's causal masking. With causal masking, the layer update for step $t+1$ can only leverage previous steps $\{0, \dots, t+1\}$.

DLL introduces the self-purification module $Q_\ell(\cdot)$ for each layer $\ell \in \{0, \dots, L\}$, anchoring the layer-wise latent targets to the observable target x_{t+1} with the self-purification loss $\mathcal{L}_{\text{pur}}$:

$$\mathcal{L}_{\text{pur}} = \frac{1}{(L+1)} \sum_{\ell \in \{0,1,\dots,L\}} \text{loss}\Big(Q_\ell\Big(\text{sg}\Big(h_{t+1}^{(\ell)}\Big)\Big), x_{t+1}\Big), \qquad \ell \in \{0, 1, \dots, L\}, \tag{3}$$

where $\text{sg}(\cdot)$ is the stop-gradient operator that prevents $\mathcal{L}_{\text{pur}}$ from updating the hidden states $h_{t+1}^{(\ell)}, \ell \in \{0, 1, \dots, L\}$.

3.3 Deep Latent Targets Prediction

For each $\ell \in \{0, \dots, L\}$, the deep latent prediction module $P_\ell(\cdot)$ maps the final-layer hidden state $h_t^{(L)}$ at the current step t to predict the self-purified latent targets $Q_\ell\Big(h_{t+1}^{(\ell)}\Big), \ell \in \{0, 1, \dots, L\}$ at the next step, supervised by the latent loss $\mathcal{L}_{\text{lat}}$:

$$\mathcal{L}_{\text{lat}} = \frac{1}{(L+1)} \sum_{\ell \in \{0,1,\dots,L\}} \text{loss}\Big(P_\ell\Big(h_t^{(L)}\Big), \text{sg}\Big(Q_\ell\Big(h_{t+1}^{(\ell)}\Big)\Big)\Big), \qquad \ell \in \{0, 1, \dots, L\}, \tag{4}$$

where $Q_\ell\Big(h_{t+1}^{(\ell)}\Big)$ is the self-purified latent target of layer ℓ at the next step $t+1$. The $\text{sg}(\cdot)$ stop-gradient operator prevents $\mathcal{L}_{\text{lat}}$ from updating the self-purified latent targets.

3.4 Training Objective

The final training objective is:

$$\mathcal{L} = \mathcal{L}_{\text{obs}} + \mathcal{L}_{\text{lat}} + \mathcal{L}_{\text{pur}}, \tag{5}$$

with the observable loss $\mathcal{L}_{\text{obs}}$, the latent loss $\mathcal{L}_{\text{lat}}$, and the self-purification loss $\mathcal{L}_{\text{pur}}$. In most cases, the loss functions are consistent at the same scale.

DLL-specific loss terms, namely $\mathcal{L}_{\text{lat}}$ and $\mathcal{L}_{\text{pur}}$, are applied only during training and introduce no inference-time overhead.

4 Experiments

We evaluate DLL across three domains with established benchmarks:

- **Computer Vision**: In Section 4.1, we follow standard protocols (Bao et al., 2022; Xu et al., 2025; He et al., 2022) to pre-train DLL from scratch and fine-tune it on ImageNet-1K (Russakovsky et al., 2015) for image classification, comparing its performance against canonical models like MAE (He et al., 2022), JEPA (Assran et al., 2023), and NEPA (Xu et al., 2025).
- **Natural Language Processing**: In Sections 4.2 and 4.3, we apply DLL to LoRA (Hu et al., 2021) fine-tune Qwen3-4B-Instruct-2507 (Yang et al., 2025) and evaluate on TableBench (Wu et al., 2025), CTI-MCQ and CTI-RCM (Alam et al., 2024), respectively. DLL is evaluated against the ablation baseline trained using only the observable loss $\mathcal{L}_{\text{obs}}$, namely cross-entropy.
- **Time Series Forecasting**: In Section 4.4, we post-train time series foundation models Toto-2.0-313m and Toto-2.0-2.5B, and evaluate on the GIFT-Eval benchmark (Aksu et al., 2024). DLL is compared against the ablation baseline trained using only the observable loss $\mathcal{L}_{\text{obs}}$ (i.e. pinball loss). In Section 4.5, we train DLL of different backbones from scratch on Chronos datasets (Ansari et al., 2024) and TSlib (Wu et al., 2023), evaluating them on the test splits against the corresponding $\mathcal{L}_{\text{obs}}$-only ablation baseline.

4.1 IMAGENET-1K CLASSIFICATION

We evaluate DLL on the ImageNet-1K dataset, adopting an autoregressive paradigm with a standard Vision Transformer (ViT) backbone (Dosovitskiy et al., 2021) and causal attention masking. Images are divided into patches, converted into a sequence of patch embeddings. As described in Section 3, both the inputs and the targets share a same network to generate representations of patches. Following a similar experimental setting to NEPA (Xu et al., 2025), we set the patch size to 14×14 and the batch size to 4096, adopt an exponential moving average (EMA (Tarvainen & Valpola, 2017)) model, and pre-train for 1600 and 800 epochs on ViT-Base and ViT-Large, respectively (detailed configurations in Appendix B.1). We first pre-train from scratch on the ImageNet-1K dataset without labels, then we fine-tune it on the labeled ImageNet-1K dataset. The classification head is simply a linear layer, the latent prediction module $P_\ell(\cdot)$ and the self-purification module $Q_\ell(\cdot)$ are both MLPs. The loss terms of DLL ($\mathcal{L}_{\text{obs}}, \mathcal{L}_{\text{lat}}$, and $\mathcal{L}_{\text{pur}}$) are uniformly formulated as negative cosine similarity in pre-training and cross-entropy in fine-tuning (following NEPA (Xu et al., 2025)).

Results are reported in Table 1. DLL achieves the highest Top-1 accuracy on ImageNet-1K classification compared to established models such as NEPA (Xu et al., 2025), MAE (He et al., 2022), and JEPA (Assran et al., 2023), reaching **83.9%** on ViT-Base and **86.0%** on ViT-Large. In Appendix C.4, we also provide a 2-norm condition number analysis of representation collapse against NEPA-Base (Xu et al., 2025; Jing et al., 2022).

Table 1: **Top-1 accuracy comparison of self-supervised models on ImageNet-1K classification.** All non-DLL results (marked by *) are directly taken from NEPA (Xu et al., 2025), including its own method and its reproduced baselines. Results are grouped by the backbone scale into ViT-B and ViT-L. **Bold** indicates the best accuracy within each backbone scale.

Model	Pre-train Task	Decoder	Epochs	Acc (%)
ViT-B				
BEiT-B* (Bao et al., 2022)	masked token pred	linear pred. head	800	83.4
DINO-B* (Caron et al., 2021)	self-distillation	mlp proj. head	1600	83.6
MAE-B* (He et al., 2022)	masked pixel pred	transformer decoder	1600	83.6
NEPA-B* (Xu et al., 2025)	autoreg. embed pred	none	1600	83.8
DLL-B	autoreg. latent & pixel pred	none	1600	**83.9**
ViT-L				
iBot-L* (Zhou et al., 2022)	self-dist & masked token pred	mlp proj. head	1000	84.8
BEiT-L* (Bao et al., 2022)	masked token pred	linear pred. head	800	85.2
JEPA-L* (Assran et al., 2023)	masked embed pred	transformer predictor	300	85.2
NEPA-L* (Xu et al., 2025)	autoreg. embed pred	none	800	85.3
MAE-L* (He et al., 2022)	masked pixel pred	transformer decoder	1600	85.6
DLL-L	autoreg. latent & pixel pred	none	800	**86.0**

4.2 LORA FINE-TUNING ON TABLEBENCH

We apply DLL to LoRA (Hu et al., 2021) fine-tune Qwen3-4B-Instruct-2507 (Yang et al., 2025) on the TableInstruct dataset and evaluate on the TableBench benchmark (Wu et al., 2025). We compare DLL against DLL-baseline trained using only the observable loss $\mathcal{L}_{\text{obs}}$, namely cross-entropy. The training and evaluation settings are listed in Appendix B.2. As shown in Table 2, the untuned model achieves an **overall mixed score** (Wu et al., 2025) of 17.90; DLL-baseline scores 25.61, whereas DLL reaches 26.41, representing an additional improvement of 0.80 points (a 3.1% relative improvement). Two representative task categories illustrate the additional improvements:

- **Numerical reasoning:** DLL reaches an exact-match (EM) score of 18.89%, compared with 18.14% for DLL-baseline and 10.83% for the untuned model. While DLL-baseline improves EM by 7.31 percentage points over the untuned model, DLL yields an additional 0.75 percentage points improvement.
- **Data analysis:** DLL achieves a score of 24.41, whereas DLL-baseline reaches 23.24 and the untuned model scores 15.86. Therefore, DLL extends the 7.38 points improvement from DLL-baseline by a further 1.17 points.

Fact-checking EM is close for DLL-baseline and DLL (78.12% and 78.22%, respectively), whereas visualization Pass@1 decreases from 34.00% with DLL-baseline to 32.00% with DLL, a decline of 2.00 percentage points for DLL.

Table 2: **TableBench results.** Comparison of the untuned Qwen3-4B-Instruct-2507 (Yang et al., 2025), DLL-baseline fine-tuned, and DLL fine-tuned. Higher is better; **bold** marks the best result in each column.

Model	Fact-checking EM (%)	Numerical EM (%)	Data analysis Mixed score	Visualization Pass@1 (%)	Overall Mixed score
Qwen3-4B-Instruct	63.54	10.83	15.86	**38.00**	17.90
Qwen3-4B-Instruct w/ DLL-baseline	78.12	18.14	23.24	34.00	25.61
Qwen3-4B-Instruct w/ DLL	**78.22**	**18.89**	**24.41**	32.00	**26.41**

4.3 LoRA Fine-tuning on CTI-MCQ and CTI-RCM

We also use DLL to fine-tune Qwen3-4B-Instruct-2507 (Yang et al., 2025) with LoRA (Hu et al., 2021) for cybersecurity evaluation on CTI-MCQ (2,500 questions) and CTI-RCM (1,000 questions) from CTI-Bench (Alam et al., 2024). CTI-MCQ assesses cybersecurity knowledge through multiple-choice questions, while CTI-RCM evaluates vulnerability classification by mapping vulnerability descriptions to Common Weakness Enumeration (CWE) categories (Alam et al., 2024). DLL-baseline uses only the observable loss $\mathcal{L}_{\text{obs}}$, implemented as cross-entropy, whereas DLL retains this objective and adds latent supervision $\mathcal{L}_{\text{lat}}$ and self-purification loss $\mathcal{L}_{\text{pur}}$. We follow the zero-shot instruction-following protocol used in Foundation-Sec (Kassianik et al., 2025) and compare DLL against DLL-baseline. The training and evaluation settings are likewise listed in Appendix B.2. The reported $\pm$ values denote standard deviations across repeated evaluations. As shown in Table 3, DLL achieves higher accuracy than DLL-baseline on both tasks:

- **CTI-MCQ:** DLL reaches $60.55\% \pm 0.41\%$, whereas DLL-baseline reaches $58.68\% \pm 0.29\%$. This corresponds to a 1.87 additional percentage points increase from DLL-baseline to DLL.
- **CTI-RCM:** DLL reaches $69.14\% \pm 0.51\%$, outperforming DLL-baseline ($66.64\% \pm 0.23\%$) by 2.50 percentage points.

Table 3: **CTI-MCQ and CTI-RCM results.** Accuracy (%) on CTI-MCQ and CTI-RCM, reported as mean $\pm$ sample standard deviation over five repeated evaluations of each model. Higher is better; **bold** marks the highest mean in each task. The untuned model and DLL-baseline follow the CyberSecQwen-4B (Mulia, 2026) implementation and evaluation.

Model	CTI-MCQ Accuracy (%)	CTI-RCM Accuracy (%)
Qwen3-4B-Instruct	42.89 ± 1.70	51.98 ± 0.34
Qwen3-4B-Instruct w/ DLL-baseline	58.68 ± 0.29	66.64 ± 0.23
Qwen3-4B-Instruct w/ DLL	$\mathbf{60.55 \pm 0.41}$	$\mathbf{69.14 \pm 0.51}$

4.4 Post-Training Evaluation on GIFT-Eval

We use DLL to post-train the decoder-only time series foundation models, Toto-2.0-313m and Toto-2.0-2.5B (Khwaja et al., 2026), with 13 datasets from *GiftEval Pre-training datasets* (Aksu et al., 2024) and *Chronos datasets* (Ansari et al., 2024) (details listed in Appendix B.4). The loss terms of DLL ($\mathcal{L}_{\text{obs}}, \mathcal{L}_{\text{lat}}$, and $\mathcal{L}_{\text{pur}}$) uniformly inherit the pinball loss from the pre-training stage of Toto-2.0 (Khwaja et al., 2026). The DLL-baseline variant is implemented under the same setting as DLL, but is post-trained using only the observable loss $\mathcal{L}_{\text{obs}}$ to serve as an ablation baseline. The post-training configurations are listed in Appendix B.3. As shown in Table 4, post-training with DLL consistently improves both MASE and CRPS across both model scales (313m and 2.5B) on the GIFT-Eval benchmark:

- **MASE performance in Table 4a:** On *Toto-2.0-313m* (Khwaja et al., 2026), DLL reduces MASE from 0.703 to 0.700, approaching the 0.699 MASE of the $\sim 3\times$ larger *Toto-2.0-1B* base model (Khwaja et al., 2026), while DLL-baseline only reaches an MASE of 0.702. On *Toto-2.0-2.5B* (Khwaja et al., 2026), DLL reduces MASE from 0.696 to 0.694, approaching *Timer-S1* (Liu et al., 2026) and *tafsut* (Tafsut-FM, 2026) (0.693 MASE), the immediate next-ranked entries on the leaderboard, whereas DLL-baseline shows no improvement.
- **CRPS performance in Table 4b:** On *Toto-2.0-313m* (Khwaja et al., 2026), DLL reduces CRPS from 0.481 to 0.480, while DLL-baseline yields no gain over the base model. On *Toto-2.0-2.5B* (Khwaja et al., 2026), DLL further reduces CRPS from 0.476 down to 0.473, surpassing the next entry *t0-beta* (0.474) (The Forecasting Company, 2026), whereas DLL-baseline only reaches 0.475.

Table 4: **Post-training performance on the GIFT-Eval benchmark.** The non-DLL results (marked by *) correspond to a contiguous sequence of entries on the current GIFT-Eval leaderboard (Salesforce AI Research, n.d.), starting from Toto-2.0-313m. Gray indicates the original Toto-2.0-313m and Toto-2.0-2.5B on the leaderboard, and **Blue** shows the results of these Toto-2.0 models post-trained with DLL and DLL-baseline.

(a) **MASE performance.** Lower MASE indicates better performance.

Model	MASE
Toto-2.0-313m*	0.703
Toto-2.0-313m w/ DLL-baseline	0.702
Toto-2.0-313m w/ DLL	0.700
Toto-2.0-1B*	0.699
Chronos-2*	0.698
Toto-2.0-2.5B*	0.696
Toto-2.0-2.5B w/ DLL-baseline	0.696
Toto-2.0-2.5B w/ DLL	0.694
Timer-S1*	0.693
tafsut*	0.693

(b) **CRPS performance.** Lower CRPS indicates better performance.

Model	CRPS
Toto-2.0-313m*	0.481
Toto-2.0-313m w/ DLL-baseline	0.481
tafsut*	0.481
Toto-2.0-313m w/ DLL	0.480
Toto-2.0-1B*	0.478
Zeus*	0.478
Toto-2.0-2.5B*	0.476
Toto-2.0-2.5B w/ DLL-baseline	0.475
t0-beta*	0.474
Toto-2.0-2.5B w/ DLL	0.473

4.5 Cross-Backbone Evaluation on Chronos Datasets and TSlib

We evaluate DLL with extensive Transformer backbones and experimental settings on Chronos datasets (Ansari et al., 2024) and TSlib (Wu et al., 2023). All experiments adopt an 80:20 train-test split. As an ablation control, the DLL-baseline variant uses the exact same configuration as DLL but is trained solely with the observable loss $\mathcal{L}_{\text{obs}}$. We report *improvement (%)* as:

$$\frac{E_{\text{base}} - E_{\text{DLL}}}{E_{\text{base}}} \times 100\%,$$

where E_{base} and E_{DLL} denote the corresponding error of the DLL-baseline and DLL. Larger values indicate stronger error improvement.

As reported in Table 5, DLL achieves superior performance than DLL-baseline:

- **Cross-backbone training from scratch:** On decoder-only, decoder-MoE and encoder-decoder backbones, DLL is implemented in the *decoder* part as described in Section 3. We adopt PatchTST (Nie et al., 2023) as the encoder-only implementation. With evaluation on Chronos datasets (Ansari et al., 2024) and TSlib (Wu et al., 2023), DLL achieves relative error improvements ranging from 0.64% to 2.20%, compared to DLL-baseline. On the 30 sub-datasets experiment with decoder-only backbone, DLL obtains 22 wins, 4 ties, and 4 losses (per-dataset results are in Appendix C.1).
- **TimesFM 2.5 post-training:** We post-train TimesFM 2.5 (Das et al., 2024), a decoder-only time series foundation model, on Chronos datasets (Ansari et al., 2024). Under the same post-training setup, DLL-baseline reaches a best test loss of 0.2306, while DLL reduces it to 0.2282, corresponding to an error improvement of 1.04%.

Table 5: **Comparative evaluation of DLL against DLL-baseline.** DLL is implemented with various Transformer backbones on Chronos datasets (Ansari et al., 2024) and TSlib (Wu et al., 2023). The encoder-only architecture adopts PatchTST (Nie et al., 2023). Additionally, TimesFM 2.5 (Das et al., 2024) post-training serves as a supplementary experiment to evaluate foundation model post-training performance on Chronos datasets (Ansari et al., 2024).

Setting / Backbone	Dataset source	Metric	**DLL-baseline**	**DLL**	Improvement (%)
decoder-only	Chronos datasets	MSE	0.7055	0.6981	1.05
decoder-MoE	Chronos datasets	MSE	0.7069	0.6971	1.38
encoder-decoder	Chronos datasets	MSE	0.5527	0.5468	1.06
decoder-only	30 Chronos sub-datasets	avg. MSE	0.5066	0.4954	2.20
encoder-only (PatchTST)	4 TSLib sub-datasets	avg. MSE	0.2647	0.2630	0.64
TimesFM 2.5	Chronos datasets	MAE	0.2306	0.2282	1.04

4.6 Ablation Studies on Chronos Datasets

We conduct ablation studies on Chronos datasets (Ansari et al., 2024) with decoder-only backbone, specifically key component ablation and hyperparameter ablation:

Key Component Ablation. Table 6 validates the key components of DLL. Removing the self-purification loss $\mathcal{L}_{\text{pur}}$ degrades MSE/MAE from $0.6981/0.1457$ to $0.7047/0.1482$ (+0.95%/+1.72% relative error), confirming its critical role in anchoring latent representations. Restricting supervision to the first layer increases MSE/MAE to $0.7036/0.1480$ (+0.79%/+1.58%), proving that deep layer supervision is essential. Replacing the latent modules with simple fully connected linear layers (FC-Q and FC-P) causes negligible performance drops (+0.11%/+0.21% and +0.11%/+0.27%), indicating that gains stem primarily from deep latent targets rather than complex module designs. Finally, adding a slow-update target network slightly hurts full DLL (MSE/MAE to $0.6989/0.1458$) but stabilizes the shallow-only variant (improving MSE from 0.7036 to 0.7031), suggesting that deep layer supervision inherently stabilizes training without needing slow updates. Visualization is provided in Appendix C.3.

Table 6: **Ablation study on key components.** Performance is evaluated as MSE and MAE. W/o Pur. removes $\mathcal{L}_{\text{pur}}$; Shallow uses only the first-layer latent target; Shallow + EMA and w/ EMA add a slow-update target branch; FC-Q/FC-P replace the purifier/predictor modules with fully connected linear layers.

Metric	**Full DLL**	w/o Pur.	Shallow	Shallow + EMA	FC-Q	FC-P	w/ EMA
MSE	**0.6981**	0.7047	0.7036	0.7031	0.6989	0.6989	0.6989
MAE	**0.1457**	0.1482	0.1480	0.1475	0.1460	0.1461	0.1458

Hyperparameter Ablation. We control one axis at a time: model depth with latent-supervised layers, the number of training data slices, and embedding size.

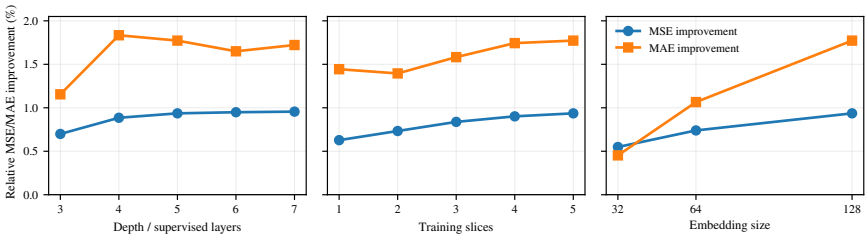


Figure 2: **Ablation on hyperparameters.** Error improvement (%) of DLL against DLL-baseline is reported across different depths, training slices, and embedding sizes. Each group changes one axis while keeping the remaining setting fixed, and vertical axis is the test MSE/MAE improvement (%).

As Figure 2 reveals, DLL yields persistent error improvement (%) as the model becomes deeper, the number of training slices increases, or the embedding size increases. Detailed numerical values are in Appendix C.2.

5 Computational and Memory Efficiency of Training

We evaluate the training computational and memory efficiency of DLL against DLL-baseline on two models of different scales and tasks: post-training Toto-2.0-313m as in Section 4.4 and LoRA fine-tuning Qwen3-4B-Instruct-2507 as in Section 4.2. In both settings, DLL-baseline is trained with the observable loss $\mathcal{L}_{\text{obs}}$ only, whereas DLL adds layer-wise latent supervision $\mathcal{L}_{\text{lat}}$ and self-purification loss $\mathcal{L}_{\text{pur}}$ over a fraction r of the backbone layers per training step (the *layer sample rate*; r=1.0 supervises every layer by default). Meanwhile, DLL induces no inference-time computational costs.

We measure **FLOPs/step** in a single forward pass over a cached training batch. **Peak Memory** is the maximum per-device memory allocated during training. DLL trades a modest compute and memory overhead for improved task performance over DLL-baseline in both settings, and r controls this trade-off:

- **Overhead of full DLL:** At r=1.0, DLL adds about 51–52% FLOPs/step and 28–31% peak memory over DLL-baseline on both models, while improving task performance: on Toto-2.0-313m, MASE decreases from 0.7022 to 0.7001 and CRPS from 0.4814 to 0.4802; on Qwen3-4B-Instruct-2507, the overall mixed score rises from 25.61 to 26.41.
- **Layer subsampling savings:** Lowering r from 1.0 to 0.3 cuts FLOPs/step by roughly 23% and peak memory by 10–11% on both models, at a marginal performance cost (Toto-2.0-313m: MASE from 0.7001 to 0.7005, CRPS from 0.4802 to 0.4803; Qwen3-4B: overall score from 26.41 to 26.33), preserving most of DLL's improvements while leaving only about **16%** FLOPs/step and **13–18%** peak memory of overhead over DLL-baseline.

Appendix A presents numerical details and reports the per-step FLOPs, peak training memory, and task performance of each variant.

6 Limitations

Lightweight Architecture Design. The current implementation of DLL is intentionally lightweight: the latent prediction module $P_\ell(\cdot)$ and self-purification module $Q_\ell(\cdot)$ primarily rely on simple MLPs. While this design minimizes computational complexity, more expressive auxiliary modules (e.g., attention layers) might better capture complex transitions across deep latent spaces.

Uniform Layer Supervision and Weighting. In our current formulation, every layer $\ell \in \{0, \ldots, L\}$ contributes equally to $\mathcal{L}_{\text{lat}}$ and $\mathcal{L}_{\text{pur}}$ with a fixed weight $1/(L+1)$. In practice, representation dynamics evolve non-uniformly across network depth—shallow layers capture local surface features while deep layers encode semantic context. The current study does not exhaustively optimize layer-selective supervision (e.g., supervising only a subset of deep layers) or dynamic loss weighting schemes.

7 Conclusion

This paper introduced **Deep Latent Learning (DLL)**, a novel training paradigm for autoregressive architectures. DLL leverages contextualized representations in deep network layers as auxiliary supervisory targets. By training the network to predict both future observables and their corresponding layer-wise deep latent states, DLL encourages models to capture rich sequence-wide context without sacrificing granular detail. Extensive evaluations demonstrate that DLL yields consistent performance improvements over established models and DLL-baseline across diverse domains, backbones, and training regimes—all while introducing zero computational cost at inference time.

AI USE STATEMENT

We used generative AI tools to polish the language and improve the clarity of the manuscript. We did not use generative AI tools for producing research ideas, experimental results, data, or code. We have reviewed all AI-assisted text. We take responsibility for the final content of this work.

ETHICS STATEMENT

This work studies a general training method evaluated on image classification, table reasoning, cybersecurity, and time-series forecasting benchmarks. The cybersecurity experiments assess security knowledge and vulnerability classification. Improved security-related capabilities may have dual-use implications, and model outputs may contain errors or biases inherited from the underlying data and pretrained models. Applications involving security decisions therefore require independent validation and human oversight. Use of the underlying datasets and pretrained models remains subject to their respective licenses and access conditions.

REPRODUCIBILITY STATEMENT

Section 3 specifies the DLL architecture and training objectives, and Section 4 describes the evaluation protocols. Appendices B.1, B.2, and B.3 provide training and evaluation configurations, while Appendix B.4 lists the Toto-2.0 post-training datasets. Per-dataset results, detailed ablations, and efficiency measurements are provided in Appendices C.1, C.2, and A, respectively.

REFERENCES

Taha Aksu, Gerald Woo, Juncheng Liu, Xu Liu, Chenghao Liu, Silvio Savarese, Caiming Xiong, and Doyen Sahoo. Gift-eval: A benchmark for general time series forecasting model evaluation, 2024. URL https://arxiv.org/abs/2410.10393.

Md Tanvirul Alam, Dipkamal Bhusal, Le Nguyen, and Nidhi Rastogi. Ctibench: A benchmark for evaluating llms in cyber threat intelligence. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 50805–50825. Curran Associates, Inc., 2024. doi: 10.52202/079017-1607. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5acd3c628aa1819fbf07c39ef73e7285-Paper-Datasets_and_Benchmarks_Track.pdf.

Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, Jasper Zschiegner, Danielle C. Maddix, Hao Wang, Michael W. Mahoney, Kari Torkkola, Andrew Gordon Wilson, Michael Bohlke-Schneider, and Yuyang Wang. Chronos: Learning the language of time series, 2024. URL https://arxiv.org/abs/2403.07815.

Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15619–15629, June 2023.

Alexei Baevski, Wei-Ning Hsu, Qiantong Xu, Arun Babu, Jiatao Gu, and Michael Auli. data2vec: A general framework for self-supervised learning in speech, vision and language. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 1298–1312. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/baevski22a.html.

Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. BEiT: BERT pre-training of image transformers. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=p-BhZSz59o4.

Adrien Bardes, Jean Ponce, and Yann LeCun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. In *ICLR*, 2022.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9650–9660, October 2021.

Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15750–15758, June 2021.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 10148–10167. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/das24c.html.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423/.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

Kawin Ethayarajh. How contextual are contextualized word representations? Comparing the geometry of BERT, ELMo, and GPT-2 embeddings. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 55–65, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1006. URL https://aclanthology.org/D19-1006/.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, koray kavukcuoglu, Remi Munos, and Michal Valko. Bootstrap your own latent - a new approach to self-supervised learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 21271–21284. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/f3ada80d5c4ee70142b17b8192b2958e-Paper.pdf.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16000–16009, June 2022.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Li Jing, Pascal Vincent, Yann LeCun, and Yuandong Tian. Understanding dimensional collapse in contrastive self-supervised learning, 2022. URL https://arxiv.org/abs/2110.09348.

Paul Kassianik, Baturay Saglam, Alexander Chen, Blaine Nelson, Anu Vellore, Massimo Aufiero, Fraser Burch, Dhruv Kedia, Avi Zohary, Sajana Weerawardhena, Aman Priyanshu, Adam Swanda, Amy Chang, Hyrum Anderson, Kojin Oshiba, Omar Santos, Yaron Singer, and Amin Karbasi. Llama-3.1-foundationai-securityllm-base-8b technical report, 2025. URL https://arxiv.org/abs/2504.21039.

Emaad Khwaja, Chris Lettieri, Gerald Woo, Eden Belouadah, Marc Cenac, Guillaume Jarry, Enguerrand Paquin, Xunyi Zhao, Viktoriya Zhukov, Othmane Abou-Amal, Chenghao Liu, Ameet Talwalkar, and David Asker. Toto 2.0: Time series forecasting enters the scaling era, 2026. URL https://arxiv.org/abs/2605.20119.

Yann LeCun. A path towards autonomous machine intelligence, 2022. URL https://openreview.net/forum?id=BZ5a1r-kVsf. Version 0.9.2, 2022-06-27. https://openreview.net/forum?id=BZ5a1r-kVsf.

Yong Liu, Xingjian Su, Shiyu Wang, Haoran Zhang, Haixuan Liu, Yuxuan Wang, Zhou Ye, Yang Xiang, Jianmin Wang, and Mingsheng Long. Timer-s1: A billion-scale time series foundation model with serial scaling, 2026. URL https://arxiv.org/abs/2603.04791.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

Samuel Mulia. Cybersecqwen-4b: A compact cti specialist fine-tuned from qwen3-4b-instruct-2507 on amd mi300x, 2026. URL https://huggingface.co/athena129/CyberSecQwen-4B.

Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *International Conference on Learning Representations*, 2023.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models, 2020. URL https://arxiv.org/abs/1910.02054.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, April 2015. ISSN 1573-1405. doi: 10.1007/s11263-015-0816-y. URL http://dx.doi.org/10.1007/s11263-015-0816-y.

Salesforce AI Research. GIFT-Eval Leaderboard. https://huggingface.co/spaces/Salesforce/GIFT-Eval, n.d. Accessed via the official project repository on 2026-09-23.

Tafsut-FM. Tafsut Univariate Base. https://huggingface.co/Tafsut-FM/tafsut-univariate-base, 2026. Model and software release; accessed 2026-09-23.

Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/68053af2923e00204c3ca7c6a3150cf7-Paper.pdf.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4593–4601, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1452. URL https://aclanthology.org/P19-1452/.

The Forecasting Company. t0: A time-series forecasting foundation model, 2026. URL https://huggingface.co/theforecastingcompany/t0-beta.

Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. Timesnet: Temporal 2d-variation modeling for general time series analysis. In *International Conference on Learning Representations*, 2023.

Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xeron Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, Tongliang Li, Zhoujun Li, and Guanglin Niu. Tablebench: A comprehensive and complex benchmark for table question answering. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(24):25497–25506, April 2025. ISSN 2159-5399. doi: 10.1609/aaai.v39i24.34739. URL http://dx.doi.org/10.1609/aaai.v39i24.34739.

Sihan Xu, Ziqiao Ma, Wenhao Chai, Xuweiyi Chen, Weiyang Jin, Joyce Chai, Saining Xie, and Stella X. Yu. Next-embedding prediction makes strong vision learners, 2025. URL https://arxiv.org/abs/2512.16922.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Jinghao Zhou, Chen Wei, Huiyu Wang, Wei Shen, Cihang Xie, Alan Yuille, and Tao Kong. ibot: Image bert pre-training with online tokenizer. *International Conference on Learning Representations (ICLR)*, 2022.

A Additional Results of Efficiency

Table 7 details the FLOPs, memory, and performance trade-off across the layer sample rate r for both settings. Decreasing r from 1.0 to 0.3 reduces FLOPs and peak memory substantially, while task performance degrades slightly, confirming that layer subsampling preserves most of the gain at a fraction of the overhead (Section 5).

Table 7: **Computational and memory efficiency of DLL.** Results are grouped by different experiments, post-training on Toto-2.0-313m and LoRA fine-tuning on Qwen3-4B-Instruct-2507. Per-step forward FLOPs and per-device peak training memory, as long as task performance, are evaluated across the layer sample rate r (r=1.0 supervises every backbone layer; DLL-baseline uses the observable loss only). Lower is better for FLOPs, memory, MASE, and CRPS; while higher is better for the overall mixed score. **Bold** marks the best task performance within each backbone. Different from Section 4.4, MASE and CRPS are rounded to 4 decimal places for more precise comparisons.

Toto-2.0-313m (*GIFT-Eval*)					
Model	**Layer Sample Rate r**	**FLOPs/step (TFLOPs)**	**Peak Memory (GB)**	**MASE**	**CRPS**
DLL-baseline	none	0.456	21.10	0.7022	0.4814
DLL	0.3	0.531	24.84	0.7005	0.4803
DLL	0.5	0.569	25.49	0.7004	0.4804
DLL	1.0	0.692	27.63	**0.7001**	**0.4802**
Qwen3-4B-Instruct-2507 (*TableBench*)					
Model	**Layer Sample Rate r**	**FLOPs/step (TFLOPs)**	**Peak Memory (GB)**	**Overall Mixed score**	
DLL-baseline	none	42.8	18.01	25.61	
DLL	0.3	49.5	20.40	26.33	
DLL	0.5	53.9	21.25	26.34	
DLL	1.0	64.6	23.04	**26.41**	

B Experimental Settings

B.1 ImageNet-1K Pre-training Configurations

Table 8 lists the ImageNet-1K pre-training configurations of DLL-Base and DLL-Large in Section 4.1, with most settings following NEPA (Xu et al., 2025). The two variants share the patch size (14×14), total batch size (4096), optimizer (AdamW), precision (bf16), and the ReLU activation of the DLL MLP modules, and differ only in the model backbone (ViT-Base vs. ViT-Large) and the number of training epochs (1600 vs. 800).

Table 8: **Pre-training configurations for DLL-Base and DLL-Large on ImageNet-1K.** Most configurations follow NEPA (Xu et al., 2025). DLL MLP activation is the activation function of self-purification module $Q_\ell(\cdot)$ and deep latent prediction module $P_\ell(\cdot)$.

Hyperparameter	DLL-Base	DLL-Large
Model backbone	ViT-Base	ViT-Large
Learning rate	3×10^{-4}	3×10^{-4}
Patch size	14×14	14×14
Total batch size	4096	4096
Training epochs	1600	800
Optimizer	AdamW	AdamW
Precision	bf16	bf16
EMA decay rate	0.9999	0.9999
DLL MLP activation	ReLU	ReLU

B.2 QWEN3-4B-INSTRUCT-2507 POST-TRAINING CONFIGURATIONS

Training Settings. Table 9 summarizes the training settings for the TableBench and CyberSec (CTI-MCQ and CTI-RCM (Alam et al., 2024)) experiments as in Section 4.2 and 4.3. Both use Qwen3-4B-Instruct-2507 (Yang et al., 2025) with LoRA (Hu et al., 2021) applied to the attention query, key, value, and output projections and the MLP gate, up, and down projections.

Table 9: **Training settings of Qwen3-4B-Instruct-2507 post-training.** Configurations of the runs used for the TableBench and CyberSec results. The CyberSec configuration is inherited from CyberSecQwen-4B (Mulia, 2026).

Parameter	TableBench	CyberSec
Training epochs	1	10
Learning rate	5×10^{-6}	5×10^{-5}
LoRA rank / alpha	64 / 64	64 / 64
LoRA dropout	0.05	0.05
Max. training tokens	10,240	4,096
Precision	bf16	bf16

For TableBench (Wu et al., 2025), both models are fine-tuned on TableInstruct (Wu et al., 2025). CyberSec (Alam et al., 2024) uses the project's vulnerability-classification and synthetic cybersecurity question-answering corpora. Both DLL runs use DLL-specific sampled supervision at Transformer layers $4, 8, 12, 16, 20, 24, 28, 32$, and 36.

Evaluation Settings. Table 10 lists the evaluation settings used for the untuned model, DLL-baseline, and DLL. TableBench (Wu et al., 2025) evaluation covers all 886 questions and uses the official task-specific metrics and overall mixed score. For CTI-MCQ and CTI-RCM from CTI-Bench (Alam et al., 2024), accuracy uses all questions as the denominator, with unparseable answers counted as incorrect. We report the mean and sample standard deviation over five repeated evaluations of each fixed model.

Table 10: **Evaluation settings of Qwen3-4B-Instruct-2507 post-training.** Within each task, the listed settings apply to the untuned model, DLL-baseline, and DLL. SD denotes sample standard deviation across repeated evaluations of the same model. CTI-MCQ and CTI-RCM settings are inherited from CyberSecQwen-4B (Mulia, 2026)

Parameter	TableBench	CTI-MCQ	CTI-RCM
Questions per evaluation	886	2,500	1,000
Input format	Chat	Chat	Chat
Prompt protocol	DP	IFT	IFT
Decoding	Greedy	Sampling	Sampling
Temperature	N/A	0.3	0.3
Max. generated tokens	8,000	512	512
Evaluations per model	1	5	5
Aggregation	Single-run score	Mean $\pm$ SD	Mean $\pm$ SD

DP denotes direct prompting; IFT denotes the instruction-following chat protocol. Temperature is not applicable to the greedy TableBench runs.

B.3 TOTO-2.0 POST-TRAINING CONFIGURATIONS

Table 11 lists the key post-training configurations of Toto-2.0-313m and Toto-2.0-2.5B (Khwaja et al., 2026) used in Section 4.4, shared by DLL and DLL-baseline.

Both models are post-trained for 2 epochs in fp32 precision with a patch size of 32 and a context length of 192 (i.e., 32×6 patches).

Toto-2.0-313m is trained with DDP, while Toto-2.0-2.5B adopts DeepSpeed ZeRO Stage 2 (Rajbhandari et al., 2020).

Table 11: **Key post-training configurations for Toto-2.0-313m and Toto-2.0-2.5B.**

Hyperparameter	Toto2-313m	Toto2-2.5B
Pretrained model	toto-2.0-313m	toto-2.0-2.5b
Learning rate	4×10^{-5}	4×10^{-5}
Strategy	DDP	DeepSpeed ZeRO Stage 2
Precision	fp32	fp32
Patch size	32	32
Context length	192 (32×6)	192 (32×6)
Epochs	2	2
DLL MLP activation	ReLU	ReLU

B.4 TOTO-2.0 POST-TRAINING DATASETS

Table 12 lists the 13 datasets from Chronos datasets (Ansari et al., 2024) and GIFT-Eval Pre-training datasets (Aksu et al., 2024) that are used to post-train Toto-2.0-313m and Toto-2.0-2.5B in Section 4.4.

Table 12: **Toto-2.0 post-training datasets.** The 13 datasets from Chronos datasets (Ansari et al., 2024) and GIFT-Eval Pre-training datasets (Aksu et al., 2024).

Dataset	Frequency
m4_hourly	Hourly
m4_daily	Daily
m4_weekly	Weekly
m4_monthly	Monthly
m4_quarterly	Quarterly
electricity_15min	15-min
monash_kdd_cup_2018	Hourly
monash_covid_deaths	Daily
monash_m1_quarterly	Quarterly
monash_saugeenday	Daily
monash_temperature_rain	Daily
fred_md	Monthly
nn5_daily_with_missing	Daily

C SUPPLEMENTARY EXPERIMENTAL RESULTS

C.1 DECODER-ONLY 30-SUB-DATASET IMPROVEMENT DISTRIBUTION

Figure 3 details the per-dataset results underlying the win/tie/loss summary reported in Section 4.5: for each of the 30 decoder-only sub-dataset pairs, it plots the relative test MSE improvement of DLL over DLL-baseline, computed as defined in Section 4.5. Improvements within ± 0.1 percentage points are treated as ties, yielding 22 wins, 4 ties, and 4 losses. The per-dataset numerical values are listed in Table 13.

C.2 DETAILS OF HYPERPARAMETER ABLATION

Table 14 reports the detailed numerical results of hyperparameter ablation in Section 4.6. Each group changes one axis while keeping the remaining settings fixed.

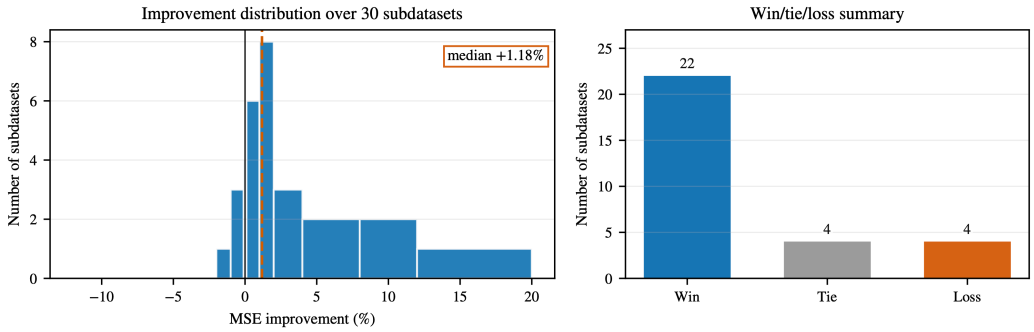


Figure 3: **Improvement distribution and win/tie/loss summary over 30 sub-datasets.** The left panel reports the relative test MSE improvement for each sub-dataset, and the right panel summarizes the number of wins, ties, and losses. DLL improves on most sub-datasets, indicating that the aggregate improvement is not driven by a single sub-dataset.

Table 13: **Decoder-only paired test MSE results over the 30 sub-datasets**. Relative improvement is computed as $(E_{\text{base}} - E_{\text{DLL}})/E_{\text{base}} \times 100\%$ within each dataset. Following the main-paper win/tie/loss threshold, changes within ± 0.1 percentage points are treated as ties. The last row reports the average DLL-baseline MSE, average DLL MSE, and the relative percentage point improvement of the two averages.

Dataset	**DLL MSE**	**DLL-Baseline MSE**	Improvement (%)	Outcome
dominick	0.5898	0.6152	4.14	Win
ercot	0.7347	0.9021	18.56	Win
exchange_rate	0.3809	0.3842	0.84	Win
m4_daily	0.0583	0.0587	0.68	Win
m4_monthly	0.1678	0.1731	3.07	Win
m4_yearly	0.5942	0.5967	0.42	Win
m5	0.6409	0.6509	1.55	Win
mexico_city_bikes	0.4926	0.4929	0.07	Tie
monash_car_parts	0.7818	0.7824	0.09	Tie
monash_cif_2016	0.5594	0.5695	1.78	Win
monash_covid_deaths	0.0421	0.0462	8.93	Win
monash_electricity_hourly	0.0513	0.0510	-0.59	Loss
monash_electricity_weekly	0.0742	0.0750	1.10	Win
monash_hospital	0.1890	0.1888	-0.09	Tie
monash_kdd_cup_2018	1.9658	1.9671	0.07	Tie
monash_london_smart_meters	0.7399	0.7452	0.71	Win
monash_m1_yearly	1.3608	1.3713	0.77	Win
monash_m3_monthly	0.3988	0.4051	1.54	Win
monash_m3_quarterly	0.5699	0.5635	-1.14	Loss
monash_m3_yearly	0.8053	0.8156	1.26	Win
monash_nn5_weekly	0.3570	0.3579	0.25	Win
monash_tourism_monthly	0.3024	0.3137	3.58	Win
monash_tourism_quarterly	1.0349	1.0535	1.76	Win
monash_traffic	0.3835	0.3822	-0.32	Loss
nn5	0.3325	0.3389	1.89	Win
taxi_1h	0.3129	0.3119	-0.35	Loss
taxi_30min	0.3526	0.3618	2.55	Win
uber_tlc_daily	0.0806	0.0844	4.54	Win
weatherbench_daily	0.2596	0.2857	9.14	Win
weatherbench_weekly	0.2496	0.2529	1.30	Win
Average	**0.4954**	**0.5066**	**2.20**	**22 W / 4 T / 4 L**

Table 14: **Hyperparameter ablation on decoder-only backbone.** Each group changes one axis while keeping other settings fixed. For the depth axis, the number of latent-supervised layers increases with model depth. **Bold** indicates the results of default setting.

Controlled axis	Value	Baseline MSE	w/ DLL MSE	Baseline MAE	w/ DLL MAE
Depth / supervised layers	3	0.7072	0.7022	0.1493	0.1476
	4	0.7072	0.7009	0.1495	0.1468
	5 (default)	**0.7055**	**0.6981**	**0.1484**	**0.1457**
	6	0.7060	0.6993	0.1484	0.1460
	7	0.7059	0.6991	0.1481	0.1456
Training data slices	1	0.7186	0.7141	0.1546	0.1524
	2	0.7101	0.7049	0.1506	0.1485
	3	0.7073	0.7014	0.1494	0.1470
	4	0.7059	0.6995	0.1487	0.1461
	5 (default)	**0.7055**	**0.6981**	**0.1484**	**0.1457**
Embedding sizes	32	0.7275	0.7235	0.1600	0.1593
	64	0.7171	0.7118	0.1545	0.1528
	128 (default)	**0.7055**	**0.6981**	**0.1484**	**0.1457**

C.3 Visualization of Key Component Ablation

To complement the numerical results of key component ablation in Section 4.6, Figure 4 visualizes the relative error increases over full DLL in percentage points. Farther right indicates stronger degradation from full DLL.

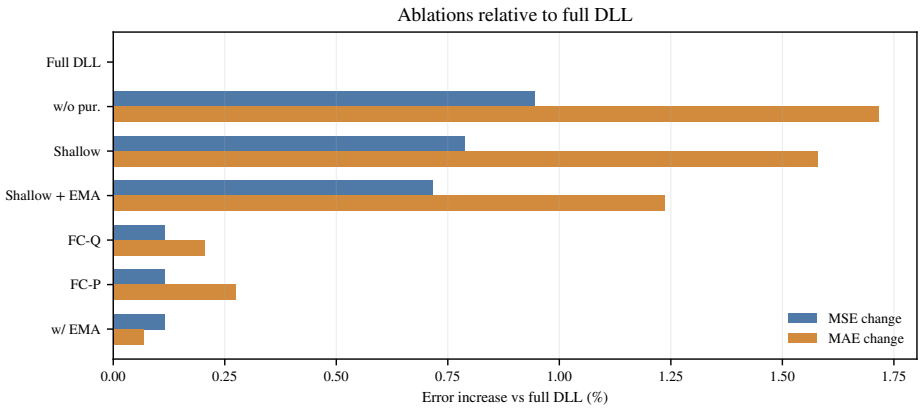


Figure 4: **Key components ablation visualization.** The horizontal axis is the percentage point increase of relative error over full DLL.

C.4 Condition Number Analysis

We compare NEPA-Base (Xu et al., 2025) and DLL-Base through the 2-norm condition number of the patch embedding projection during training. We flatten the projection weights into a matrix W, with one row per output channel, and compute

$$\kappa_2(W) = \frac{\sigma_{\max}(W)}{\sigma_{\min}(W)},$$

where $\sigma_{\max}(W)$ and $\sigma_{\min}(W)$ are the largest and smallest singular values, respectively. This ratio measures the relative scaling of the most amplified and most compressed input directions; larger values may suggest a greater risk of representation collapse (Jing et al., 2022).

As shown in Figure 5, DLL-Base maintains a substantially lower κ_2 after the initial rise. NEPA-Base (Xu et al., 2025) remains around 10,000–12,000 for much of training, with large fluctuations,

whereas DLL-Base shows an overall decline after its early peak. Near the end of training, DLL-Base reaches approximately 1,800, compared with 7,800 for NEPA-Base (Xu et al., 2025).

The lower κ_2 for DLL-Base indicates a smaller ratio between the extreme singular values and less severe compression relative to the most amplified direction. These results indicate substantially better spectral conditioning under DLL-Base, consistent with alleviated representation collapse at the patch embedding projection (Jing et al., 2022).

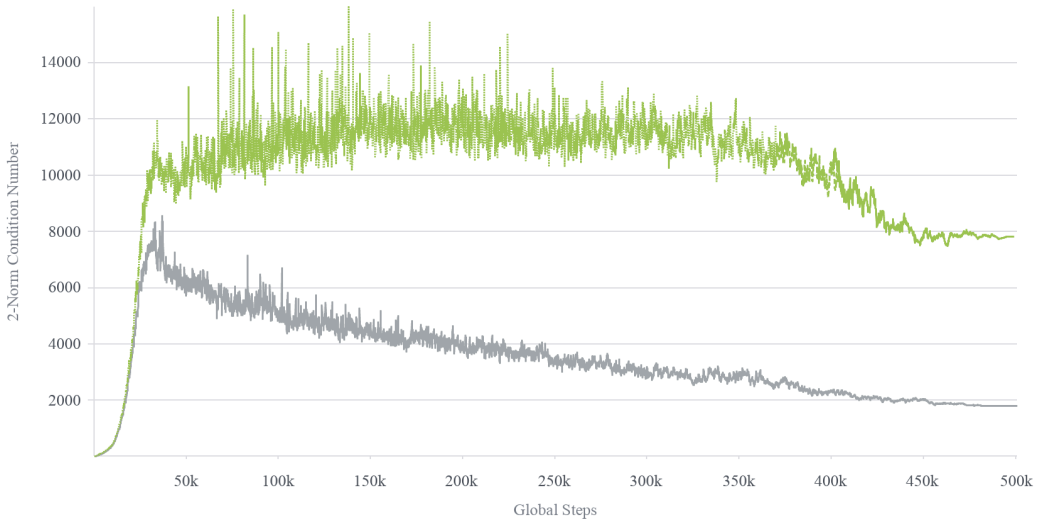


Figure 5: **Spectral conditioning of the patch embedding projection.** The 2-norm condition number κ_2 of the flattened projection weight matrix is plotted against training steps for **NEPA-Base (green)** and **DLL-Base (gray)**. After the initial rise, DLL-Base maintains a lower condition number and shows an overall decline during training.