Efficient Autoregressive Inference for Tabular Foundation Models

Conor Hassan[1,2], **Nasrulloh Loka**[3], **Cen-You Li**[3], **Daolang Huang**[1,2], **Paul E. Chang**[3,4], **Yang Yang**[3], **Francesco Silvestrin**[3], **Samuel Kaski**[1,2,5], **Luigi Acerbi**[3]

[1]ELLIS Institute Finland
[2]Department of Computer Science, Aalto University, Finland
[3]Department of Computer Science, University of Helsinki, Finland
[4]DataCrunch
[5]Department of Computer Science, University of Manchester, UK
conor.hassan@aalto.fi

Abstract

Transformer-based tabular foundation models excel at single-pass *marginal* prediction, yet many applications require coherent *joint distributions* across predictions. Purely autoregressive architectures capture dependencies but forgo flexible set-conditioning used in meta-learning; deploying set-based models autoregressively forces re-encoding the augmented context at each step. We introduce a *causal autoregressive buffer* that encodes the context once, caches it, and uses a causal buffer for generated targets. Targets attend to the cache and the visible buffer prefix, enabling efficient batched autoregressive generation and one-pass joint log-likelihoods. A unified training scheme (masked attention with a buffer-size curriculum) covers both modes with minimal overhead. On a small tabular foundation model, the buffer matches joint estimates from existing approaches while delivering up to 20$\times$ faster joint sampling.

1 Introduction

Set-based conditioning models—neural processes and transformer variants (Garnelo et al., 2018a; Nguyen & Grover, 2022; Chang et al., 2025), prior-fitted networks (Müller et al., 2022), and recent tabular foundation models (Hollmann et al., 2023, 2025; Jingang et al., 2025)—summarize variable-sized *context sets* with permutation-invariant encoders, enabling rapid marginal prediction on new targets. Many applications, however, need joint distributions over multiple targets to capture dependencies. A common approach deploys these models autoregressively (Bruinsma et al., 2023): after each prediction, the conditioning set is augmented and re-encoded, breaking caching and yielding a bottleneck of $\mathcal{O}(K(N+K)^2)$ for K targets and context size N. Efficient attention (Jaegle et al., 2021; Feng et al., 2023) helps for static contexts but not repeated recomputation.

We propose a *causal autoregressive buffer* that decouples one-time context encoding from sequential dependence modeling. The context $\mathcal{C}$ is encoded once and cached; a dynamic buffer stores previously predicted targets. New targets attend to both the cache and the causal buffer, eliminating repeated context passes and enabling parallelism. A unified training strategy using masked attention and a buffer-size curriculum lets a single model deliver efficient marginal predictions and accelerated autoregressive sampling and likelihood evaluation with minimal extra training cost and comparable accuracy to standard AR deployment.

Our main contributions are: (i) A causal autoregressive buffer that separates set-based context encoding from sequential dependence, enabling efficient joint sampling and likelihoods. (ii) A uni-

AI for Tabular Data workshop at EurIPS 2025.

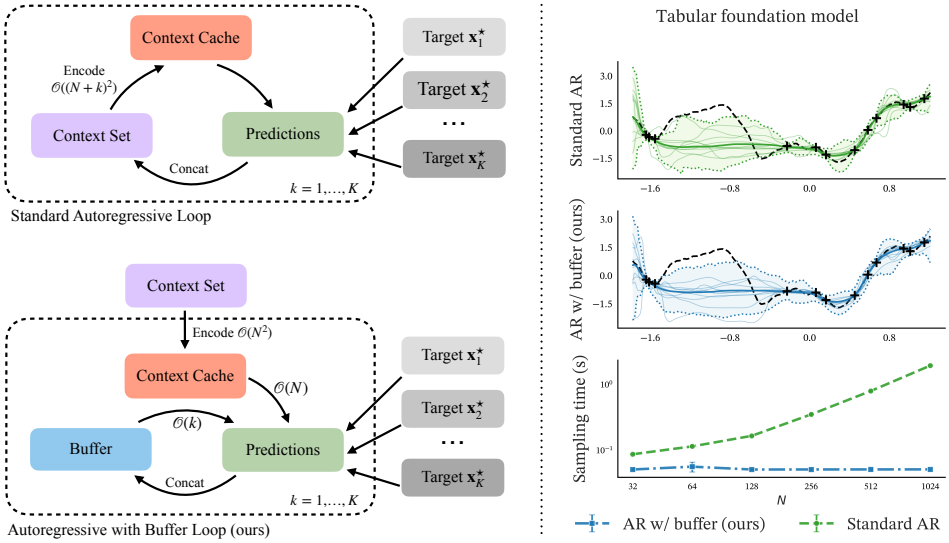

Figure 1: Standard AR re-encodes the augmented context at each step, costing $\mathcal{O}(K(N+K)^2)$. Our buffered approach encodes $\mathcal{C}$ once and caches it; new targets enter a causal buffer that attends to the cache and prior buffer entries, avoiding re-encoding. On a tabular foundation model, accuracy matches the non-buffered baseline while achieving up to $20\times$ **faster** joint sampling.

fied training scheme (masked attention, buffer-size curriculum) that covers both modes with minimal overhead. (iii) Empirical evidence showing up to $20\times$ speedup with comparable predictive accuracy.

2 Background

We consider meta-learning tasks where a model adapts to new prediction problems from observed data without retraining. Given a *context set* $\mathcal{C} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ and a *target set* $\mathcal{T} = \{(\mathbf{x}_m^\star, y_m^\star)\}_{m=1}^M$, the goal is to learn $p_{\boldsymbol{\theta}}(y_{1:M}^\star \mid \mathbf{x}_{1:M}^\star; \mathcal{C})$ (Foong et al., 2020). When targets are processed autoregressively we use index k for readability. Set-based transformer models such as *transformer neural processes* (TNPs; Nguyen & Grover, 2022) and *prior-fitted networks* (PFNs; Müller et al., 2022) summarize $\mathcal{C}$ with multi-head self-attention (MHSA) and decode each target $\mathbf{x}_m^\star$ using multi-head cross-attention (MHCA), producing a diagonal factorization:

$$p_{\boldsymbol{\theta}}(y_{1:M}^\star \mid \mathbf{x}_{1:M}^\star; \mathcal{C}) \;=\; \prod_{m=1}^{M} p_{\boldsymbol{\theta}}(y_m^\star \mid \mathbf{r}_{\text{tgt}}(\mathbf{x}_m^\star, \mathbf{r}_{\mathcal{C}}(\mathcal{C}))) \,. \tag{1}$$

Here $\mathbf{r}_{\mathcal{C}}(\mathcal{C})$ is a permutation-invariant context summary (via MHSA) and $\mathbf{r}_{\text{tgt}}$ is the decoder producing a parametric predictive distribution for $y_m^\star$ (e.g., a Gaussian; more expressive heads include Riemannian distributions (Müller et al., 2022) and mixtures of Gaussians (Uria et al., 2016; Chang et al., 2025)). Models are trained by maximum likelihood on random context–target splits.

Many applications require dependencies across targets (joint sampling or joint likelihood evaluation). While one can equip Eq. (1) with multivariate parametrics (Nguyen & Grover, 2022), a flexible alternative is an autoregressive (AR) factorization (Bruinsma et al., 2023):

$$p_{\boldsymbol{\theta}}(y_{1:K}^\star \mid \mathbf{x}_{1:K}^\star; \mathcal{C}) = \prod_{k=1}^{K} p_{\boldsymbol{\theta}}\left(y_k^\star \mid \mathbf{x}_k^\star;\, \mathcal{C} \cup \{(\mathbf{x}_j^\star, y_j^\star)\}_{j=1}^{k-1}\right) . \tag{2}$$

This is a deployment mode of the same set-conditioned model: each prediction conditions on all previous targets. For likelihoods, AR is order-dependent; averaging over permutations approximates permutation invariance. The drawback is computational: each augmentation of the conditioning set forces recomputation of $\mathbf{r}_{\mathcal{C}}(\cdot)$, yielding $\mathcal{O}(K(N+K)^2)$ cost and making parallel AR sampling cumbersome (e.g., B parallel sequences often require B copies of the model).

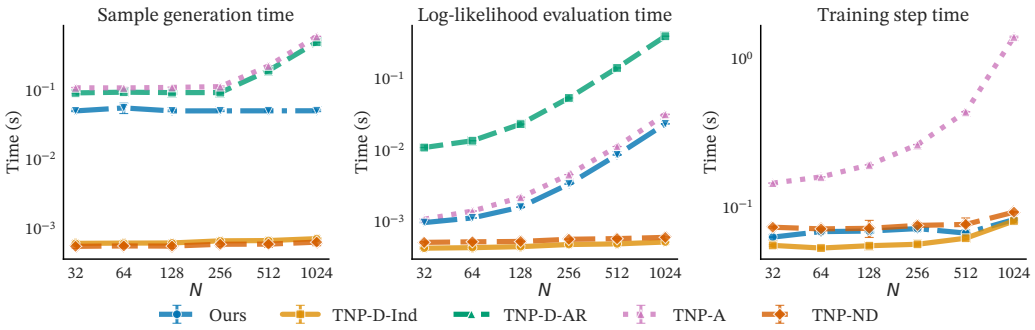

Figure 2: Wall-clock time (log-scale) for sampling, joint log-likelihood, and a training step vs. context size N. Buffered inference is faster than strong AR baselines.

3 Efficient Autoregressive Inference

We condition the predictive distribution on a static *context* $\mathcal{C}$ and a dynamic *autoregressive buffer* $\mathcal{B}$:

$$p_{\boldsymbol{\theta}}(y^\star_{1:K} \mid \mathbf{x}^\star_{1:K}; \mathcal{C}) \;=\; \prod_{k=1}^{K} p_{\boldsymbol{\theta}}(y^\star_k \mid \mathbf{r}_{\text{tgt}}(\mathbf{x}^\star_k, [\mathbf{r}_{\mathcal{C}}(\mathcal{C}), \mathbf{b}_{1:k-1}]))\,, \quad \mathbf{b}_k = \mathbf{r}_{\mathcal{B}}((\mathbf{x}^\star_k, y^\star_k), [\mathbf{r}_{\mathcal{C}}(\mathcal{C}), \mathbf{b}_{1:k-1}]), \tag{3}$$

where $\mathbf{r}_{\mathcal{C}}(\mathcal{C})$ is computed once (cached), $\mathbf{r}_{\mathcal{B}}$ is a causal-MHSA encoder, and $\mathbf{b}_{1:k-1}$ are the *encoded* buffer tokens ($\mathbf{b}_{1:0} = \emptyset$). The decoder $\mathbf{r}_{\text{tgt}}$ cross-attends once to the concatenated keys/values (K/V) from the cache and the visible buffer prefix, then outputs a parametric head. With an empty buffer, Eq. (3) reduces to Eq. (1). We enforce: **(R1)** context immutable (read-only cache); **(R2)** strictly causal buffer (token j attends only to $< j$); **(R3)** one-way flow (no writes into $\mathcal{C}$); **(R4)** targets read from the cache and the visible buffer prefix. Buffered AR costs $\mathcal{O}(N^2{+}KN{+}K^2)$ (one-time context pass, total cross-reads, causal buffer self-attention) versus $\mathcal{O}(K(N{+}K)^2)$ for naïve AR that re-encodes the augmented context at each step. Architectural details appear in Appendix A.

Training details. We minimize expected NLL over tasks from $\mathcal{P}$. For each $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_{\text{tot}}}$, a random partition π yields context $\mathcal{C}$, buffer $\mathcal{B} = \{(\mathbf{x}_k, y_k)\}_{k=1}^{K}$ (random order), and targets $\mathcal{T} = \{(\mathbf{x}_m, y_m)\}_{m=1}^{M}$, with $N_{\text{tot}} = N{+}K{+}M$. We compute all target predictions in a single masked forward pass. Mask: 50% context-only; 50% context plus a random buffer prefix $\mathcal{B}_{1:v_m}$ with $v_m \sim \text{Uniform}(1, K)$ (see Appendix A.2). The training objective is:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\mathcal{D}\sim\mathcal{P}}\left[\mathbb{E}_{(\mathcal{C},\mathcal{B},\mathcal{T})\sim\pi(\cdot|\mathcal{D})}\left[-\sum_{m=1}^{M} \log p_{\boldsymbol{\theta}}(y_m \mid \mathbf{x}_m, \mathcal{C}, \mathcal{B}_{1:v_m})\right]\right], \tag{4}$$

where $\mathcal{B}_{1:v_m}$ is the *raw* buffer prefix for target m ($v_m{=}0$ means context-only). The curriculum preserves marginal quality (frequent buffer-free predictions) and teaches incorporation of buffer information (random prefixes), aligning with posterior predictives under varying conditioning sets (Müller et al., 2022; Elsemüller et al., 2024).

Summary of inference procedures. Inference uses a two-stage routine: one-time context *prefill* ($\mathcal{O}(N^2)$), followed by *predictions* interacting with the cached context. For sampling, we decode sequentially, appending each $(\mathbf{x}^\star_k, y^\star_k)$ to the buffer cache. For joint log-likelihoods, we pack buffer tokens for $(\mathbf{x}^\star_k, y^\star_k)$ and query tokens for $\mathbf{x}^\star_m$ so each query attends to $\mathcal{C}$ and $\mathcal{B}_{1:m-1}$, computing all conditionals in one pass; averaging over orders approximates permutation invariance. Batched sampling reuses the single context cache across streams. See Appendix A for algorithms, masks, and variants; related work is in Appendix B.

4 Experiments

We evaluate two axes: (i) the *efficiency of the autoregressive buffer*, such as wall-clock times for sampling, joint log-likelihood, and a full training step; and (ii) *predictive quality* on real tabular tasks

Table 1: **Average Log-likelihood (↑) results on UCI datasets with TabICL.** We evaluate our proposed AR-Buffer mechanism integrated into a TabICL foundation model against independent and standard AR baselines. Performance is measured on both interpolation (Int) and forecasting (For) tasks across three real-world datasets. Results are reported as mean and standard error over 16 randomly sampled mini-datasets ($N = 128$, $M = 32$).

	Electric Consumption		Gas Turbine		Bike Sharing	
	Int	For	Int	For	Int	For
Independent	1.60 (0.10)	1.02 (0.29)	-0.39 (0.14)	-1.16 (0.60)	1.54 (0.06)	0.97 (0.11)
Standard AR	1.63 (0.10)	1.38 (0.27)	-0.38 (0.14)	-0.75 (0.33)	1.57 (0.06)	1.21 (0.10)
AR w/ buffer ($K = 32$)	1.61 (0.10)	1.35 (0.27)	-0.38 (0.14)	-0.76 (0.33)	1.57 (0.06)	1.18 (0.10)

with a foundation-model backbone, comparing independent predictions, versus joint predictions generated autoregressively with and without the buffer.

Computational efficiency. All models share a unified codebase, matched parameter counts, and identical input/output heads (Appendix C). We benchmark four TNP variants for joint prediction: *TNP-D-Ind* (diagonal, independent), *TNP-ND* (non-diagonal multivariate Gaussian), *TNP-D-AR* (diagonal model run autoregressively via re-encoding), and *TNP-AR* (fully autoregressive with causal self-attention). We use TNP baselines because the transformer and attention pattern of *TNP-D* match the dataset-wise in-context transformer in TabICL and TabPFN (Hollmann et al., 2023), giving a controlled efficiency comparison across alternative joint-sampling mechanisms that use similar attention operations (independent, non-diagonal, AR via re-encoding, fully AR). As context, *TNP-AR* typically matches *TNP-D-AR* accuracy but is slower, while *TNP-ND* is one-pass yet tends to underfit dependencies. All runs use KV caching, FlashAttention-2 (Dao, 2023), and compilation. Figure 2 reports time vs. context size N with the same backbone as downstream experiments; buffer K=16. Sampling/likelihood: M=16, batch B=256; training: M=256, B=128. Our method is 3–20× faster than *TNP-AR*/*TNP-D-AR* for sampling, matches *TNP-AR* and is $K\times$ faster than *TNP-D-AR* for likelihoods, and trains 4–12× faster than *TNP-AR* with minimal overhead vs. *TNP-D*/*TNP-ND*. Extended sweeps are in Appendix D.

Tabular foundation model. We instantiate the buffer in a TabICL-like architecture (Jingang et al., 2025) and adapt it to regression. The architecture has two parts: (i) a set encoder that computes feature embeddings once (cached), and (ii) a dataset-wise in-context transformer whose transformer/attention match TNP-D / TabPFN (Hollmann et al., 2023). We add a structured attention mask so predictions are written to a causal buffer while the context cache remains immutable—no re-encoding during AR inference. Pre-training follows the Sstructured causal model synthetic prior (Jingang et al., 2025) on 10.24M datasets (up to 10 features, and context sizes between 8 and 1024 with buffer size K=32; each task is split into context, buffer, and targets. Further details appear in Appendix E.1. *Results.* We evaluate on three UCI time-series datasets (Electric Consumption, Gas Turbine, Bike Sharing). For each, we form 16 tasks with N=128 context and M=32 targets under interpolation (Int) and forecasting (For). Using the same backbone, we compare: *Independent* (marginals), *Standard AR* (K=1), and *AR w/ buffer* (K=32). Table 1 shows both AR modes outperform independent predictions, and *AR w/ buffer* matches *Standard AR* within s.e., indicating the buffer preserves AR dependencies while enabling efficient inference.

5 Discussion

We introduce a causal autoregressive buffer that decouples one-time context encoding from lightweight sequential updates in set-conditioned transformers. By caching context keys/values and handling target-to-target dependencies in a causal buffer, we reduce attention from $\mathcal{O}(K(N+K)^2)$ to $\mathcal{O}(N^2+NK+K^2)$. In tabular prediction, the buffer matches autoregressive baselines while achieving up to 20× faster joint sampling, with minimal extra training cost vs. standard models and up to 10× lower training cost than fully autoregressive baselines. Gains are largest when many joint samples are drawn from the same large context and moderate K. Future work includes scaling to larger target sizes and integrating the buffer into post-pretraining fine-tuning of existing tabular foundation models.

References

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block Diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, 2025.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 2020.

Wessel P Bruinsma, James Requeima, Andrew YK Foong, Jonathan Gordon, and Richard E Turner. The Gaussian neural process. In *3rd Symposium on Advances in Approximate Bayesian Inference*, 2021.

Wessel P Bruinsma, Stratis Markou, James Requeima, Andrew YK Foong, Tom R Andersson, Anna Vaughan, Anthony Buonomo, J Scott Hosking, and Richard E Turner. Autoregressive conditional neural processes. In *International Conference on Learning Representations*, 2023.

Paul E Chang, Nasrulloh Loka, Daolang Huang, Ulpu Remes, Samuel Kaski, and Luigi Acerbi. Amortized probabilistic conditioning for optimization, simulation and inference. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2025.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, 2023.

Nicola De Cao, Wilker Aziz, and Ivan Titov. Block neural autoregressive flow. In *Uncertainty in artificial intelligence*. PMLR, 2020.

Vincent Dutordoir, Alan Saul, Zoubin Ghahramani, and Fergus Simpson. Neural diffusion processes. In *International Conference on Machine Learning*. PMLR, 2023.

Lasse Elsemüller, Hans Olischläger, Marvin Schmitt, Paul-Christian Bürkner, Ullrich Koethe, and Stefan T. Radev. Sensitivity-aware amortized bayesian inference. *Transactions on Machine Learning Research*, 2024.

Leo Feng, Hossein Hajimirsadeghi, Yoshua Bengio, and Mohamed Osama Ahmed. Efficient queries transformer neural processes. In *NeurIPS 2022 Workshop on Meta-Learning*, 2022.

Leo Feng, Hossein Hajimirsadeghi, Yoshua Bengio, and Mohamed Osama Ahmed. Latent bottlenecked attentive neural processes. In *International Conference on Learning Representations*, 2023.

Leo Feng, Frederick Tung, Hossein Hajimirsadeghi, Yoshua Bengio, and Mohamed Osama Ahmed. Memory efficient neural processes via constant memory attention block. In *International Conference on Machine Learning*. PMLR, 2024.

Andrew YK Foong, Wessel P Bruinsma, Jonathan Gordon, Yann Dubois, James Requeima, and Richard E Turner. Meta-learning stationary stochastic process prediction with convolutional neural processes. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020.

Marta Garnelo, Dan Rosenbaum, Chris J Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo J Rezende, and SM Ali Eslami. Conditional neural processes. In *International Conference on Machine Learning*. PMLR, 2018a.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Ali Eslami, and Yee Whye Teh. Neural processes. In *ICML 2018 Workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018b.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*. PMLR, 2015.

Manuel Gloeckler, Michael Deistler, Christian Weilbach, Frank Wood, and Jakob H Macke. All-in-one simulation-based inference. In *International Conference on Machine Learning*. PMLR, 2024.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *International Conference on Learning Representations*, 2023.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, 2025.

Emiel Hoogeboom, Alexey A. Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models. In *International Conference on Learning Representations*, 2022.

Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural autoregressive flows. In *International Conference on Machine Learning*. PMLR, 2018.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning*. PMLR, 2021.

QU Jingang, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICL: A tabular foundation model for in-context learning on large data. In *International Conference on Machine Learning*. PMLR, 2025.

Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2016.

Jose Lara-Rangel, Nanze Chen, and Fengzhe Zhang. Exploring pseudo-token approaches in transformer neural processes. *arXiv preprint arXiv:2504.14416*, 2025.

Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *international conference on artificial intelligence and statistics*. PMLR, 2011.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set Transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*. PMLR, 2019.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023.

Sulin Liu, Peter J Ramadge, and Ryan P Adams. Generative marginalization models. In *International Conference on Machine Learning*. PMLR, 2024.

Sarthak Mittal, Niels Leif Bracher, Guillaume Lajoie, Priyank Jaini, and Marcus A Brubaker. Exploring exchangeable dataset amortization for bayesian posterior inference. In *ICML 2023 Workshop on Structured Probabilistic Inference and Generative Modeling*, 2023.

Sarthak Mittal, Niels Leif Bracher, Guillaume Lajoie, Priyank Jaini, and Marcus Brubaker. Amortized in-context Bayesian posterior estimation. *arXiv preprint arXiv:2502.06601*, 2025.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do Bayesian inference. In *International Conference on Learning Representations*, 2022.

Samuel Müller, Matthias Feurer, Noah Hollmann, and Frank Hutter. PFNs4BO: In-context learning for bayesian optimization. In *International Conference on Machine Learning*. PMLR, 2023.

Ryan L Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *International Conference on Learning Representations*, 2019.

Tung Nguyen and Aditya Grover. Transformer Neural Processes: Uncertainty-aware meta learning via sequence modeling. In *International Conference on Machine Learning*. PMLR, 2022.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017.

Massimiliano Patacchiola, Aliaksandra Shysheya, Katja Hofmann, and Richard E Turner. Transformer neural autoregressive flows. In *ICML 2024 Workshop on Structured Probabilistic Inference & Generative Modeling*, 2024.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018. URL https://openai.com/index/language-unsupervised/.

Arik Reuter, Tim GJ Rudner, Vincent Fortuin, and David Rügamer. Can transformers learn full Bayesian inference in context? *International Conference on Machine Learning*, 2025.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*. PMLR, 2015.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

Haotian Tang, Yecheng Wu, Shang Yang, Enze Xie, Junsong Chen, Junyu Chen, Zhuoyang Zhang, Han Cai, Yao Lu, and Song Han. HART: Efficient visual generation with Hybrid AutoRegressive Transformer. In *International Conference on Learning Representations*, 2025.

Benigno Uria, Iain Murray, and Hugo Larochelle. A deep and tractable density estimator. In *International Conference on Machine Learning*. PMLR, 2014.

Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *Journal of Machine Learning Research*, 17(205):1–37, 2016.

George Whittle, Juliusz Ziomek, Jacob Rawling, and Michael A Osborne. Distribution transformers: Fast approximate Bayesian inference with on-the-fly prior adaptation. *arXiv preprint arXiv:2502.02463*, 2025.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dLLM: Training-free acceleration of diffusion LLM by enabling KV cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025.

Table of Contents

A Method Details

This appendix spells out the modules used in Eq. (3), the single block-sparse attention mask that implements requirements (R1)–(R4), and the exact procedures for autoregressive sampling and one-pass joint log-likelihood evaluation.

A.1 Modules and notation

Our method uses three sets of tokens: context $\mathcal{C}$, buffer $\mathcal{B}$, and targets $\mathcal{T}$, of sizes N, K, M, respectively. Throughout this paper, let

$$\mathbf{E}_x : \mathcal{X} \to \mathbb{R}^d, \quad \mathbf{E}_y : \mathcal{Y} \to \mathbb{R}^d, \quad \mathbf{a} : \{1, \dots, K\} \to \mathbb{R}^d$$

denote learned embeddings for inputs, outputs, and buffer positions. In addition, we introduce role embeddings that indicate token type, denoted by $e^{\text{role}}_{\text{ctx}}$, $e^{\text{role}}_{\text{buf}}$, and $e^{\text{role}}_{\text{tgt}}$ for context, buffer, and target tokens, respectively.

Context encoder $\mathbf{r}_{\mathcal{C}}$. Given context pairs $\mathcal{C} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, construct context tokens: $e^{\text{ctx}}_n = \mathbf{E}_x(\mathbf{x}_n) + \mathbf{E}_y(y_n) + e^{\text{role}}_{\text{ctx}}$, process them with bidirectional MHSA (no positional embeddings), and cache per-layer keys/values:

$$\{\text{KV}^\ell_{\mathcal{C}}\}_{\ell=1}^L \;=\; \mathbf{r}_{\mathcal{C}}(\mathcal{C}) \quad \text{(computed once; immutable)}.$$

Buffer encoder $\mathbf{r}_{\mathcal{B}}$. For a buffer prefix $\mathcal{B}_{1:k} = \{(\mathbf{x}^\star_j, y^\star_j)\}_{j=1}^k$, form tokens $e^{\text{buf}}_j = \mathbf{E}_x(\mathbf{x}^\star_j) + \mathbf{E}_y(y^\star_j) + \mathbf{a}(j) + e^{\text{role}}_{\text{buf}}$, then apply *strictly causal* MHSA on $\{e^{\text{buf}}_j\}_{j \le k}$ so that each token is restricted to attend only to earlier tokens in the sequence, and in addition, each token performs cross-attention to the cached context $\{\text{KV}^\ell_{\mathcal{C}}\}$. This yields per-layer $\text{KV}^\ell_{\mathcal{B}_{1:k}}$ that we update incrementally at inference:

$$\{\text{KV}^\ell_{\mathcal{B}_{1:k}}\}_{\ell=1}^L \;=\; \mathbf{r}_{\mathcal{B}}\left(\mathcal{B}_{1:k}, \mathbf{r}_{\mathcal{C}}(\mathcal{C})\right).$$

Target decoder $\mathbf{r}_{\text{tgt}}$ and prediction head. For a target input $\mathbf{x}^\star_m$ we create a query token $e^{\text{tgt}}_m = \mathbf{E}_x(\mathbf{x}^\star_m) + e^{\text{role}}_{\text{tgt}}$. The target decoder $\mathbf{r}_{\text{tgt}}$ performs a *single cross-attention* from e^{tgt}_m to the *concatenated* keys/values of the context cache $\{\text{KV}^\ell_{\mathcal{C}}\}$ and the *visible* buffer prefix $\{\text{KV}^\ell_{\mathcal{B}_{1:v_m}}\}$, followed by normalization and an MLP:

$$\mathbf{h}_m = \mathbf{r}_{\text{tgt}}\Big(e^{\text{tgt}}_m,\ \Big[\{\text{KV}^\ell_{\mathcal{C}}\},\ \{\text{KV}^\ell_{\mathcal{B}_{1:v_m}}\}\Big]\Big), \qquad \boldsymbol{\phi}_m = \psi(\mathbf{h}_m),$$

where ψ is the distribution head (e.g., the mixture-of-Gaussian head).

A.2 Training mask that implements (R1)–(R4)

We concatenate tokens as $[\mathcal{C}, \mathcal{B}, \mathcal{T}]$ with sizes N, K, and M, respectively, and use one block-sparse attention mask consisting of the following *five* unmasked sections (everything else is masked):

(1) Self-attention within context. Context tokens attend bidirectionally to other context tokens. Context never attends to buffer or targets (context is read-only outside this block).

(2) Buffer reads context (cross-attention). Each buffer token can read (attend to) all context tokens. This lets the buffer incorporate task information from the cached context while keeping the context cache immutable.

(3) Causal self-attention within the buffer. Within the buffer itself, attention is strictly causal: a buffer token at position j can only read earlier buffer positions $< j$ (no future reads). This encodes the autoregressive dependency among realized targets.

(4) Targets read context (cross-attention). Each target query can read the entire cached context. There are no edges between targets.

(5) Targets read buffer (prefix only, cross-attention). Each target query can read only a *visible prefix* of the buffer. The visible prefix length for target m is v_m: *training (teacher forcing):* we set $v_m{=}0$ for 50% of targets and sample $v_m \sim \text{Uniform}\{1, \dots, K\}$ for the rest (the curriculum);

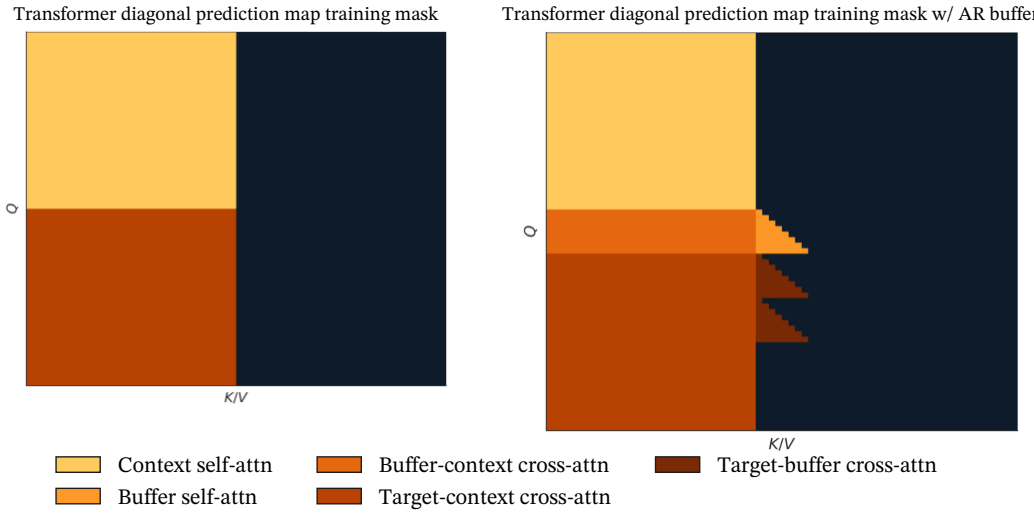

Figure A1: **Block-sparse attention masks with and without an autoregressive buffer.** *Left:* a diagonal prediction-map transformer (e.g., TNP/PFN): the context attends to itself and each target reads the entire context. *Right:* our buffered variant inserts an autoregressive memory $\mathcal{B}$ between context and targets, adding three blocks: (i) buffer reads context (ii) *causal* self-attention within buffer (iii) target reads varying number of elements from start of buffer, depending on curriculum.

sampling: at step k, the active query sees the realized prefix $k{-}1$; *one-pass joint log-likelihood:* packed queries use $v_m{=}m{-}1$ to recover the autoregressive chain in a single forward pass.

All other connections are masked: context never reads buffer or targets; targets never read targets; and buffer never reads targets. This single pattern implements the four requirements from the main text—immutable context, strictly causal buffer, unidirectional flow out of context, and target access to (context + visible buffer). See Fig. A1 for the diagram.

Complexity. Under this mask, a full prediction pass costs $\mathcal{O}(N^2{+}NK{+}K^2)$ attention operations per layer: one-time $\mathcal{O}(N^2)$ for $\mathcal{C}$, $\mathcal{O}(NK)$ for reads from $\mathcal{C}$, and $\mathcal{O}(K^2)$ for causal buffer self-attention. This replaces the $\mathcal{O}\big(K(N{+}K)^2\big)$ cost of naive AR re-encoding. Packing B target orders in parallel (for order averaging) isolates the B buffer sets while sharing the context cache, yielding $\mathcal{O}\big(N^2 + B(NK + K^2)\big)$.

A.3 Algorithms for autoregressive sampling and log-likelihood evaluation

We include here the pseudocode for the main procedures used in our method.

Autoregressive sampling. Given a context $\mathcal{C}$ and a sequence of target inputs $\mathbf{x}_1^\star, \dots, \mathbf{x}_K^\star$, we generate samples by first performing a one-time *prefill* of $\mathcal{C}$, caching its keys and values in an $\mathcal{O}(N^2)$ operation. We then *decode sequentially* following Eq. (3): for each step $k = 1, \dots, K$, we form a target query for input $\mathbf{x}_k^\star$, attend to the cached context and causal buffer $\mathcal{B}_{k-1}$, sample $y_k^\star$ from the predictive distribution, and append $(\mathbf{x}_k^\star, y_k^\star)$ to the buffer with its positional embedding. Only the buffer's key/value cache is incrementally updated, avoiding context recomputation and yielding $\mathcal{O}(N^2 + NK + K^2)$ total complexity. Algorithm 1 details the autoregressive sampling procedure.

Joint likelihood evaluation. Our framework can also evaluate the joint likelihood of a set of $K = M$ targets, $\{(\mathbf{x}_m^\star, y_m^\star)\}_{m=1}^K$, in a single forward pass. To achieve this, similar to the TNP-A variant of Nguyen & Grover (2022), we pack two sets of tokens into the model: (i) *buffer tokens* for the targets $\{(\mathbf{x}_k^\star, y_k^\star)\}_{k=1}^K$, and (ii) separate *query tokens* for the same target inputs $\{\mathbf{x}_m^\star\}_{m=1}^K$. A causal attention mask ensures that each query for $\mathbf{x}_m^\star$ attends to the context $\mathcal{C}$ and only the preceding buffer tokens $\mathcal{B}_{1:m-1} = \{(\mathbf{x}_k^\star, y_k^\star)\}_{k<m}$. This allows all conditional probabilities to be computed in one pass: $\log p_{\boldsymbol{\theta}}(y_{1:K}^\star \mid \mathbf{x}_{1:K}^\star, \mathcal{C}) = \sum_{m=1}^K \log p_{\boldsymbol{\theta}}\big(y_m^\star \mid \mathbf{x}_m^\star, \mathcal{C}, \mathcal{B}_{1:m-1}\big)$. This is algebraically

Algorithm 1 Autoregressive sample generation for K targets

Require: Context $\mathcal{C} = \{(x_n, y_n)\}_{n=1}^{N}$, target inputs $\{x_k^\star\}_{k=1}^{K}$

$\{\mathrm{KV}_{\mathcal{C}}^{\ell}\} \leftarrow \mathbf{r}_{\mathcal{C}}(\mathcal{C})$ ▷ $\mathcal{O}(N^2)$; cached

Initialize $\{\mathrm{KV}_{\mathcal{B}_{1:0}}^{\ell}\}$ ▷ empty buffer cache

for $k = 1$ to K **do**

  $\mathbf{h}_k \leftarrow \mathbf{r}_{\mathrm{tgt}}\left(\mathbf{E}_x(x_k^\star) + e_{\mathrm{tgt}}^{\mathrm{role}},\ \left[\{\mathrm{KV}_{\mathcal{C}}^{\ell}\},\ \{\mathrm{KV}_{\mathcal{B}_{1:k-1}}^{\ell}\}\right]\right)$

  Sample $y_k^\star \sim p_\theta(\cdot; \psi(\mathbf{h}_k))$

  Append $(x_k^\star, y_k^\star)$; update $\{\mathrm{KV}_{\mathcal{B}_{1:k}}^{\ell}\}$ (strictly causal)

end for

return $\{y_k^\star\}_{k=1}^{K}$

Algorithm 2 Joint log-likelihood evaluation for K targets

Require: Context $\mathcal{C} = \{(x_n, y_n)\}_{n=1}^{N}$, ordered targets $\{(x_k^\star, y_k^\star)\}_{k=1}^{K}$

$\{\mathrm{KV}_{\mathcal{C}}^{\ell}\} \leftarrow \mathbf{r}_{\mathcal{C}}(\mathcal{C})$ ▷ $\mathcal{O}(N^2)$; cached

Build all K buffer tokens; compute $\{\mathrm{KV}_{\mathcal{B}_{1:K}}^{\ell}\}$ under causal mask

Build target queries $\{\mathbf{E}_x(x_k^\star) + e_{\mathrm{tgt}}^{\mathrm{role}}\}_{k=1}^{K}$

Mask: target k sees $\mathcal{B}_{1:k-1}$ and all of $\mathcal{C}$

Compute $\{\log p_k\}_{k=1}^{K}$;

return $\sum_{k=1}^{K} \log p_k$

identical to sequential autoregressive evaluation but executes in a single forward pass with total attention cost $\mathcal{O}(N^2+KN+K^2)$. Notably, autoregressive likelihood estimates are order-dependent; to recover approximate permutation invariance, we average the likelihood over multiple buffer orderings (Murphy et al., 2019). Algorithm 2 presents the joint likelihood evaluation.

Batched autoregressive sampling. Our method is particularly efficient for autoregressively generating multiple samples in a batch, conditional on the same context $\mathcal{C}$ (e.g., multiple joint predictions for the same observed function values – see Fig. 1). A naive batched autoregressive approach must re-encode a growing context set at every generation step for each of the B samples. To generate B samples of length K, this results in a prohibitive total cost of $\mathcal{O}(BK(N+K)^2)$. In contrast, our approach performs the expensive context prefill ($\mathcal{O}(N^2)$) only *once*. This single context cache is then efficiently reused across all B batched generation streams, with only the small, dynamic buffer maintaining a separate state for each sample. This reduces the total cost to $\mathcal{O}(N^2 + B(NK + K^2))$, making batched sampling practical even for large contexts and batches.

A.4 Architectural generality

Our buffer is a general mechanism applicable to other transformer variants. For instance, a Perceiver-style encoder (Jaegle et al., 2021) summarizes the context $\mathcal{C}$ into a fixed set of $P \ll N$ latent tokens, also known as *pseudo-tokens* (Lee et al., 2019; Feng et al., 2023; Lara-Rangel et al., 2025). We can precompute the latent key/value representations once – autoregressive decoding then requires attending only to these P latents and the growing causal buffer. The per-layer attention cost is $\mathcal{O}(NP+P^2)$ for the prefill and $\mathcal{O}(PK+K^2)$ for decoding K samples. In contrast, the approach without our buffer would incur a larger cost of $\mathcal{O}(NPK+P^2K+PK^2)$.

B Related Work

Neural processes and prior-fitted networks. Our method can serve as a modular component within neural processes (NPs; Garnelo et al., 2018b,a; Bruinsma et al., 2021; Nguyen & Grover, 2022; Dutordoir et al., 2023; Chang et al., 2025) or prior-fitted networks (PFNs; Müller et al., 2022, 2023; Hollmann et al., 2023). Prior work on efficient NP methods has primarily focused on improving scalability with respect to the context set size (Feng et al., 2022, 2023) and on reducing memory usage (Feng et al., 2024) for independent prediction tasks. By contrast, our method targets

efficiency in autoregressive joint sampling and evaluation, an area that has received limited attention in the NP literature. Our contributions are complementary and can be combined with other architectural improvements.

Transformer probabilistic models. Recent work increasingly leverages transformer architectures for probabilistic modeling, framing Bayesian inference as an in-context learning task. These methods perform tasks such as approximating posterior distributions, modeling conditional relationships, and estimating posterior predictive distributions by conditioning on context observations and, optionally, additional prior information (Mittal et al., 2023; Gloeckler et al., 2024; Reuter et al., 2025; Chang et al., 2025; Whittle et al., 2025; Mittal et al., 2025). Our work builds on this direction by leveraging transformer-based variants of neural processes and prior-fitted networks.

Tabular foundation models. The effectiveness of PFNs has led to transformer-based tabular foundation models such as TabPFN (Hollmann et al., 2023, 2025) and TabICL (Jingang et al., 2025), which demonstrate strong performance on tabular data through in-context learning approaches. The "in-context learning" over *rows* within these models follows the same attention mechanisms as standard transformer neural processes and PFNs; our method integrates naturally with these models, serving as a complementary module for efficient joint sampling and prediction.

Autoregressive joint density estimation. Autoregressive approaches are widely used for joint density estimation, from neural autoregressive density estimators (Larochelle & Murray, 2011; Uria et al., 2016; Germain et al., 2015) to normalizing flows (Kingma et al., 2016; Papamakarios et al., 2017; Huang et al., 2018; De Cao et al., 2020; Patacchiola et al., 2024), and order-agnostic autoregressive models (Uria et al., 2014; Hoogeboom et al., 2022; Liu et al., 2024). Within the NP literature, our method is related to the Autoregressive Transformer NP (TNP-A; Nguyen & Grover, 2022) which duplicates targets into queries and observed values. While TNP-A uses this duplication for both training and inference, we recognize it is only needed for likelihood evaluation. Bruinsma et al. (2023) showed that deploying standard NPs autoregressively improves joint predictions but requires expensive context re-encoding at each step. Our buffer mechanism combines insights from both approaches: like TNP-A, we enable parallel likelihood evaluation, and like Bruinsma et al. (2023), we model autoregressive dependencies while training on independent targets – our separate buffer architecture avoids both TNP-A's training overhead and the re-encoding bottleneck.

Connection to other generative modeling techniques. Modern generative models for joint distributions follow two main paradigms: diffusion and flow-matching models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2021; Lipman et al., 2023) that generate samples through continuous-time dynamics, and autoregressive transformers (GPTs; Radford et al., 2018; Brown et al., 2020) that generate sequences token-by-token with cached key-value states. While diffusion dominates in continuous domains like images and video, autoregressive transformers excel in discrete sequences and show excellent performance and scalability in multiple domains. Our buffer mechanism brings the efficiency of autoregressive transformers to NPs and PFNs. Standard NPs/PFNs struggle with joint prediction because they must re-encode the entire context at each autoregressive step. Our approach instead mirrors language models: encode the set-based context once (like a prompt) and generate efficiently through cached representations. Recent work has shown these paradigms can be combined (Tang et al., 2025; Arriola et al., 2025; Wu et al., 2025), suggesting future extensions.

C Transformer Neural Process Baselines Details

We summarize the baseline transformer neural process (TNP) variants used in our comparisons, following Nguyen & Grover (2022).

C.1 TNP-D

This model takes as input a context set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ and a target set $\{\mathbf{x}_m^\star\}_{m=1}^M$. Similar to Section A, the context embeddings e_n^{ctx} are processed with bidirectional MHSA with no positional encodings. Each target is decoded by:

$$\mathbf{h}_m = \mathbf{r}_{\text{tgt}}\Big(e_m^{\text{tgt}},\ \mathbf{r}_{\mathcal{C}}(\mathcal{C})\Big), \qquad \boldsymbol{\phi}_m = \psi(\mathbf{h}_m),$$

where ψ is the distribution head (Gaussian as in the original paper; we primarily use a mixture of Gaussians). The left panel of Fig. A1 shows the training mask for TNP-D. This model is trained via maximum likelihood estimation of independent targets given a fixed context set.

At deployment, the decoding can be independent or autoregressive, yielding TNP-D-Ind and TNP-D-AR methods. TNP-D-Ind decodes all targets independently in a single pass. It is fast (context and targets encoded once), but cannot capture dependencies between targets.

TNP-D-AR decodes targets sequentially, appending each sampled $(\mathbf{x}_m^\star, y_m^\star)$ to the context. This captures joint structure but requires re-encoding the growing set at each step. TNP-D-Ind is invariant to target order; TNP-D-AR is order-sensitive, so we approximate the predictive distribution by averaging over multiple target orderings.

C.2 TNP-ND

This model encodes the context set once and decodes all targets simultaneously by parameterizing a joint multivariate Gaussian distribution over the outputs. The embedder and transformer backbone are identical to those of TNP-D-Ind:

$$\mathbf{h}_m = \mathbf{r}_{\text{tgt}}\Big(e_m^{\text{tgt}},\ \mathbf{r}_{\mathcal{C}}(\mathcal{C})\Big).$$

Then the joint distribution is obtained via

$$\boldsymbol{\phi} = \psi_{ND}(\mathbf{h}_1, \ldots, \mathbf{h}_M),$$

where ψ_{ND} is the multivariate Gaussian head that outputs both a mean vector and valid covariance matrix. The mean is produced per target, and a lightweight self-attention head over the set of targets yields fixed-width embeddings that are transformed into a valid covariance factor. This design supports a variable number of targets and is invariant to target order.

The training optimizes the joint multivariate Gaussian likelihood of the target points. At inference, the joint samples and log-likelihood are computed in a single pass. This model is invariant to the order of target points.

C.3 TNP-A

The key difference between this model and TNP-D is the attention mechanism on the target set. This model processes three sets: the context $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$, the target $\{\mathbf{x}_m^\star\}_{m=1}^M$, and the observed target $\{(\mathbf{x}_m^\star, y_m^\star)\}_{m=1}^M$. To differentiate, we denote the embeddings of $\{(\mathbf{x}_m^\star, y_m^\star)\}_{m=1}^M$ by $\{e_m^{y,\text{tgt}}\}$. Similar to TNP-D, the context embeddings attend to each other. For the target set, each e_m^{tgt} attends to the context and the previous observed target embeddings $e_{j<m}^{y,\text{tgt}}$. Likewise, the observed target embeddings attend to context and previous target embeddings (Fig. 2 of Nguyen & Grover 2022).

The target causal mask allows TNP-A to model the joint likelihood simultaneously in one single pass, assuming the observations are given (e.g., for training and test log-likelihood evaluations). For prediction generation, however, each sampled target needs to be re-encoded and attended for the generation of next targets, yielding a sequential re-encoding procedure. The causal mask on the target set is sensitive to the target order, and thus the final likelihood is an average over multiple random permutations. Note that this model processes duplicated target set–$\{\mathbf{x}_m^\star\}_{m=1}^M$ and an observed sequence $\{(\mathbf{x}_m^\star, y_m^\star)\}_{m=1}^M$; this creates significant computational overhead in both the training and the inference, particularly when the target set is large (see e.g. Section D and Figs. A7 to A9).

Compared to our method, TNP-A can be viewed as TNP-D with a '*frozen buffer*' $\{(\mathbf{x}_m^\star, y_m^\star)\}_{m=1}^M$ of size $K = M$ containing the observed targets. For likelihood evaluation where all sets are processed in one shot, the behavior of TNP-A and our approach are analogous, with the set $\{(\mathbf{x}_m^\star, y_m^\star)\}_{m=1}^M$ serving a role similar to our buffer. However, for AR sampling, TNP-A repeatedly re-encodes the full context and target sets after each sampled $y_m^\star$, whereas our method dynamically adapts to new samples. Moreover, since TNP-A does not afford a dynamic-size target set decoupled from the 'in-context' targets, training is much more expensive than our method (see Fig. 2 in the main text).

D Computational Efficiency Details

This section provides additional empirical results to support the efficiency claims made in the main paper. We present an analysis of performance scaling with batch size, an ablation study of our custom kernel, a comparison against unoptimized open-source baselines, and further ablations on training time. In all subsequent plots, the absence of a data point for a given method indicates that the experiment failed due to an out-of-memory (OOM) error for that specific configuration.

D.1 Scaling with Batch Size

To analyze the effect of batch size B, we provide expanded results for autoregressive sampling and joint log-likelihood evaluation in Fig. A2 and Fig. A3, respectively. These plots show the wall-clock time as a function of the number of context points N for various batch sizes. The results confirm that our method's performance advantage over autoregressive baselines like TNP-A is consistent and often widens as the context and batch size increase.

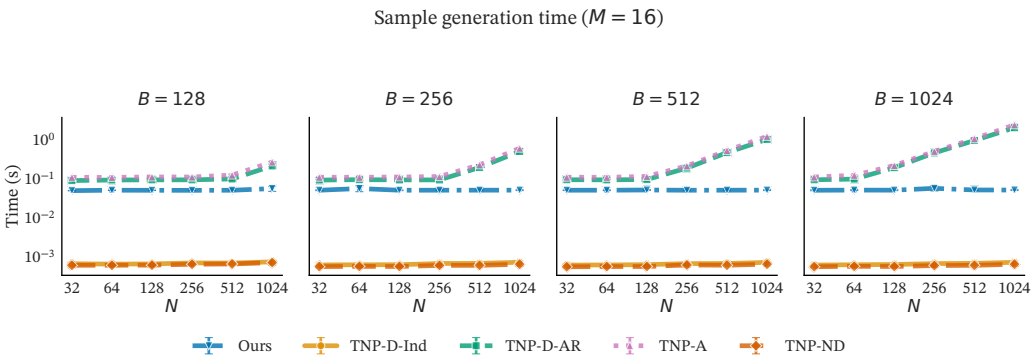

Figure A2: Autoregressive sampling time (log scale) versus context size N for an expanded range of batch sizes B.

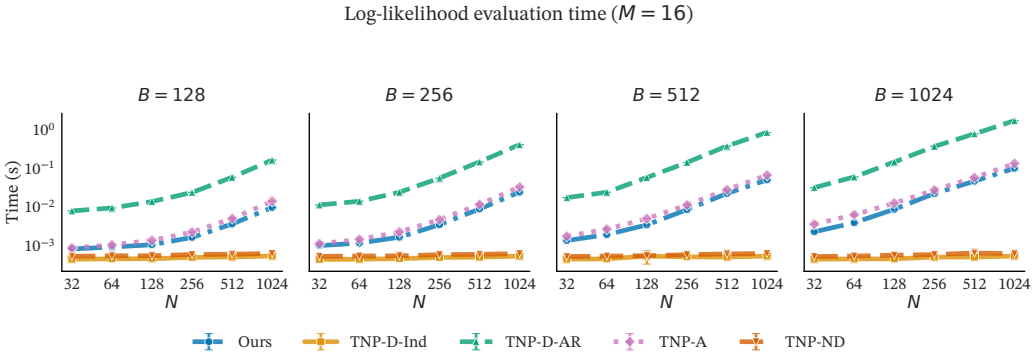

Figure A3: Joint log-likelihood evaluation time (log scale) versus context size N for an expanded range of batch sizes B.

D.2 Impact of Custom Triton Kernel

To isolate the contribution of our custom attention kernel, we compare the sampling time of our method with and without this optimization. The kernel is designed to accelerate a key computational step: the cross-attention between the batched target embeddings (batch size B) and the concatenation of a batched buffer cache with a *shared* context cache (batch size 1). A naive implementation would explicitly expand the context cache tensor B times to match the batch dimension before the attention operation. This "expand" operation is memory-bandwidth intensive and creates a large, redundant intermediate tensor.

Our custom Triton kernel avoids this bottleneck by fusing the expansion and attention computations. The kernel loads the single context cache into fast SRAM and reuses it for each item in the batch, calculating the attention on-the-fly without ever materializing the full expanded tensor in slower global memory. As shown in Fig. A4, this memory-centric optimization provides a substantial speedup that grows with the batch size B.

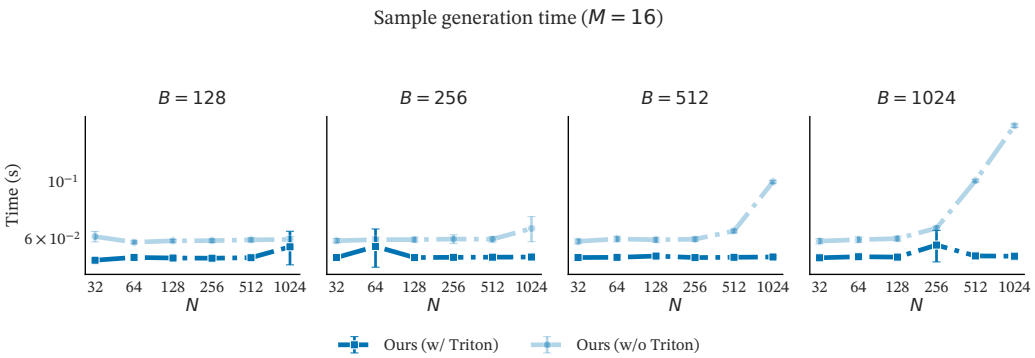

Figure A4: Ablation study for autoregressive sampling, comparing our method with and without the custom Triton kernel across different context and batch sizes.

D.3 Comparison to Open-Source Baselines

To demonstrate the fairness of our primary comparisons, we benchmark our optimized baseline implementations against their standard, publicly available versions. The results for sampling and likelihood evaluation are shown in Fig. A5 and Fig. A6. Our optimized baselines are consistently $3 - 10\times$ faster than their standard counterparts. This confirms that our method's performance gains are due to fundamental algorithmic advantages, not an unfair comparison against unoptimized code.

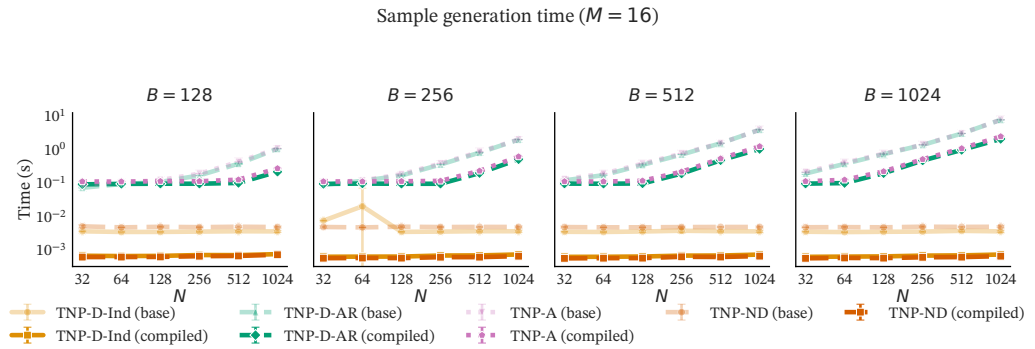

Figure A5: Comparison of our optimized baseline implementations against standard open-source versions for autoregressive sampling.

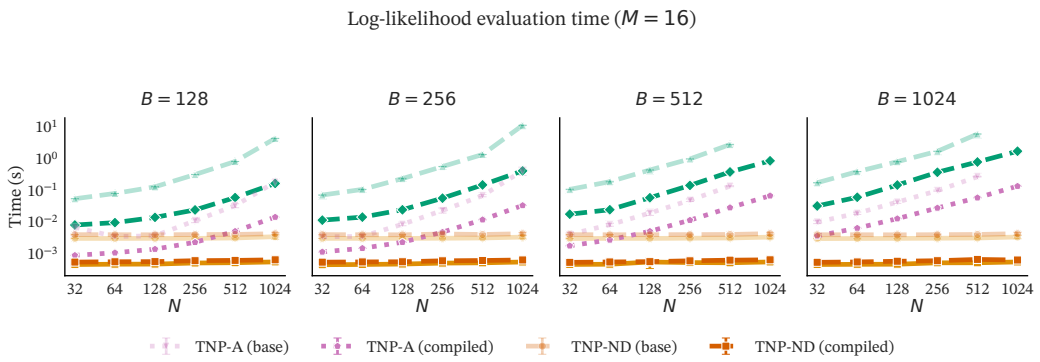

Figure A6: Comparison of our optimized baseline implementations against standard open-source versions for joint log-likelihood evaluation.

D.4 Training Time Scaling

We further analyze the scaling of training step time with respect to the number of target points M for different batch sizes. Figs. A7 to A9 show this relationship for batch sizes of 64, 128, and 256, respectively. The results show that as the context, target, or batch size increases, TNP-A becomes increasingly expensive to train relative to all other methods.

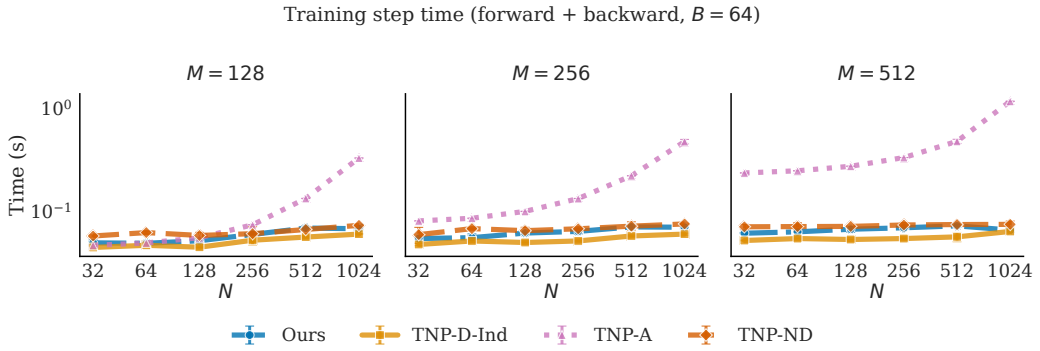

Figure A7: Training step time vs. number of target points M for batch size $B = 64$.

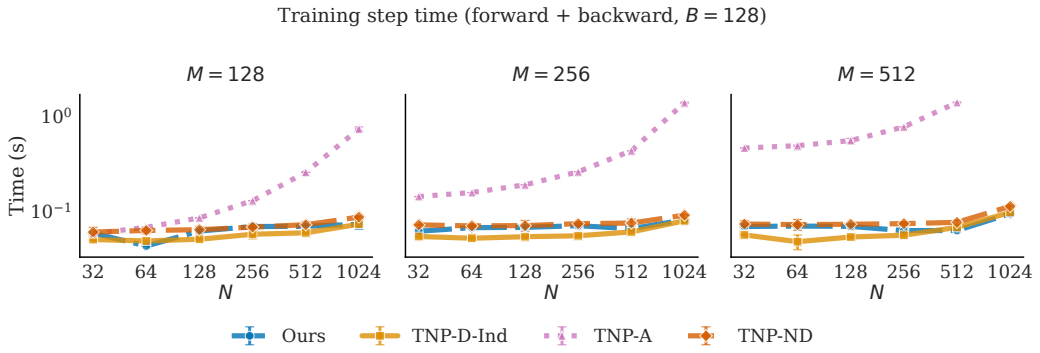

Figure A8: Training step time vs. number of target points M for batch size $B = 128$.

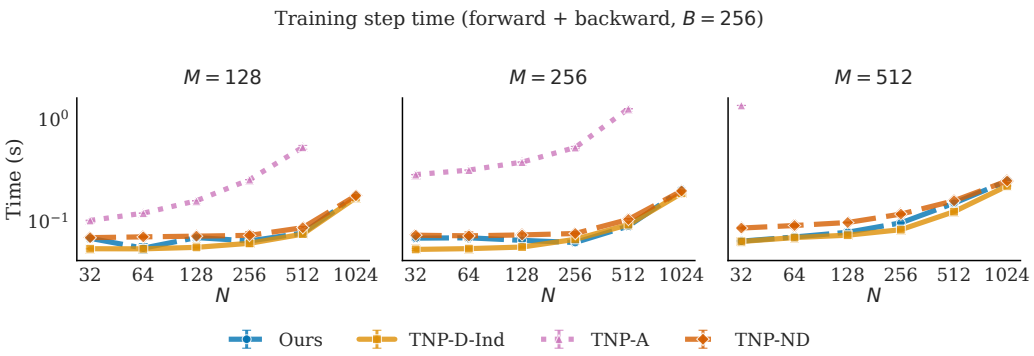

Figure A9: Training step time vs. number of target points M for batch size $B = 256$.

D.5 Impact of Attention Patterns on Training Speed

A key difference between the baseline models is their compatibility with modern, efficient attention implementations. The causal attention mask required by TNP-A during training is incompatible with kernels like FlashAttention, forcing it to use PyTorch's standard, but slower, "math" attention backend. In contrast, models like TNP-D and ours can leverage these faster kernels.

In Section C, we discussed the duplicated processing of TNP-A on the target set, which incurs significant computational overhead. To determine if TNP-A's slow training is fundamental to its architecture or merely an artifact of this kernel incompatibility, we conduct a controlled ablation. We disable FlashAttention for *all* methods, forcing a fair comparison on the same standard PyTorch attention backend. The results, shown in Figs. A10 to A12, are unequivocal. Even on a level playing field, TNP-A's training time is orders of magnitude slower than all other methods. This confirms that its high computational cost is an inherent consequence of its autoregressive design, not just an implementation detail.

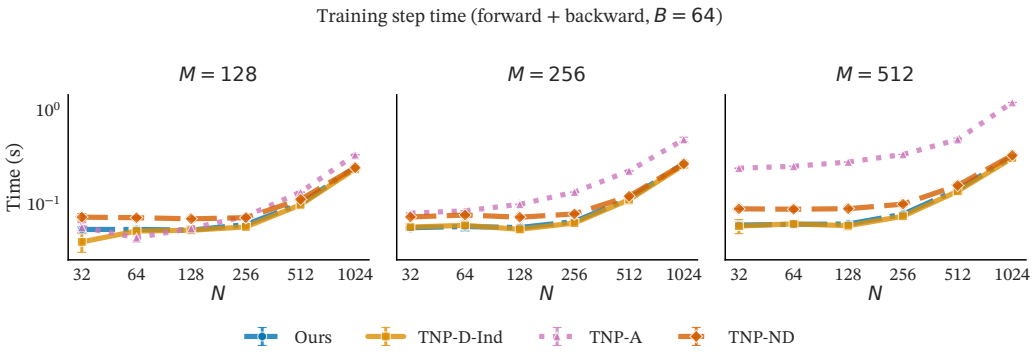

Figure A10: Training step time vs. number of target points M using the standard PyTorch attention backend (FlashAttention disabled). Batch size $B = 64$.

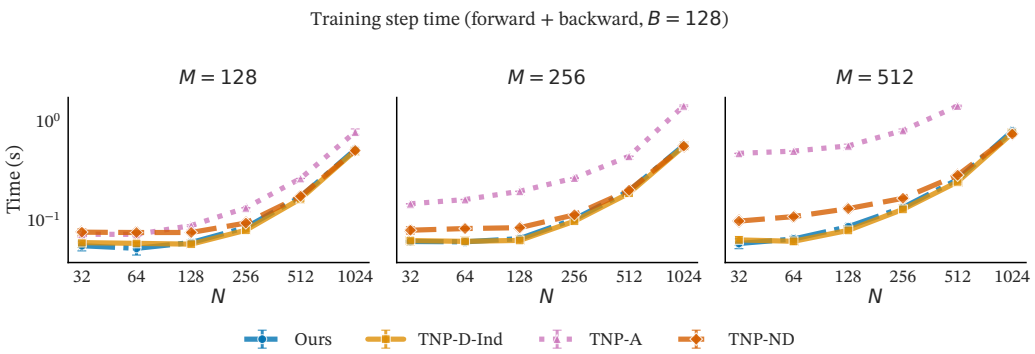

Figure A11: Training step time vs. number of target points M using the standard PyTorch attention backend (FlashAttention disabled). Batch size $B = 128$.

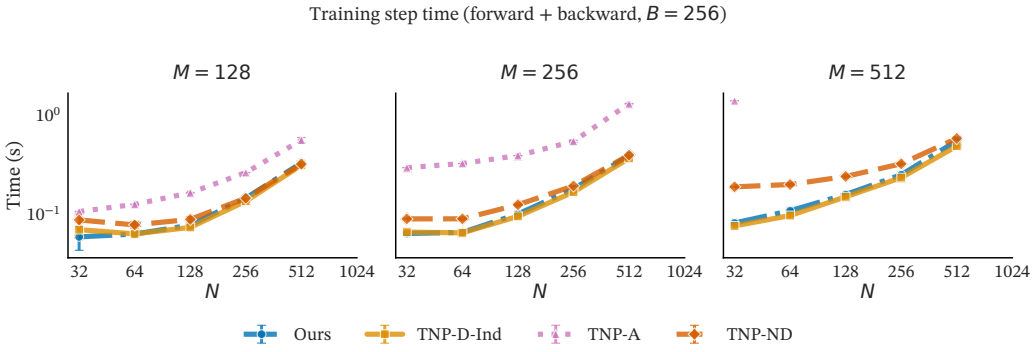

Figure A12: Training step time vs. number of target points M using the standard PyTorch attention backend (FlashAttention disabled). Batch size $B = 256$.

E Experimental Details

E.1 Tabular model details

This section describes the TabICL model and explains how the training dataset was generated. Notably, the base architecture used for this tabular data example is different from the one used in the other experiments, highlighting the broad applicability of our method.

E.1.1 Architecture

Set encoder. We reuse the first two stages of TabICL (Jingang et al., 2025) without modification: the distribution-aware column processor (TF_{col}, implemented with induced self-attention blocks) followed by the context-aware row-wise transformer (TF_{row}) with RoPE. Scalars are mapped by a $1 \to 128$ linear layer; each column is then processed across rows by an ISAB stack (Lee et al., 2019) with three blocks, four heads, 128 inducing points, feed-forward hidden dimension of 256. The row-wise encoder has three layers with four heads, feed-forward hidden dimension of 256, and RoPE base 100,000. We prepend two `[CLS]` tokens per row and concatenate their outputs, yielding a 256-dimensional row embedding (2×128). We use at most ten features per table.

Tokenization and additive target encoding. The set encoder produces one row token per sample for context, buffer, and target rows (dimension 128; only selects the subset of the vector corresponding to the `[CLS]` token dimensions). Context and buffer tokens receive the target value *additively* via a small target encoder (linear $1 \to 128$. Buffer tokens also receive a learned positional embedding indicating their autoregressive index (up to 32 positions). This keeps labels additive, lets us compute the set encoder once, and makes the buffer explicit at the token level.

Dataset-wise ICL with a buffered mask. On top of these tokens we run a dataset-wise transformer with twelve layers and four heads, model width 128, and feed-forward size 256. The attention mask is the only architectural change relative to TabICL: context attends bidirectionally and is read-only at inference; the buffer uses strictly causal self-attention; target queries attend to the cached context and to the causal prefix of the buffer; there are no edges into context from buffer or targets. The maximum buffer size is 32 tokens and we query 512 targets per task.

Prediction head. Predictions use a GMM head with 20 components and a minimum standard deviation of 10^{-3}.

Caching. The column and row set encoder is computed once for all rows. During autoregressive decoding we cache keys/values for the context once and update only the buffer cache, so the same context cache is reused across parallel generations.

E.1.2 Data generation and preprocessing

SCM prior and task family. We generate datasets with the MLP-based *structured causal model* (SCM) prior in the style of Hollmann et al. (2023), following the dataset-wise, set-encoded regime of TabICL (Jingang et al., 2025). Concretely, we first sample a DAG with layered (MLP-style) connectivity and then define each variable c as $c = f(\mathrm{Pa}(c)) + \varepsilon$, where $\mathrm{Pa}(c)$ are its parents, f is a small MLP with nonlinearity, and ε is independent noise. Unless stated otherwise, we sample the feature dimension $d \in [1, 10]$, and per-task context sizes $N \in [8, 1024]$; targets are continuous responses with dataset-specific noise levels. The cause sampler follows the TabPFN prior (including mixed marginals); the SCM therefore yields columns that may be non-Gaussian or discrete at source, which we handle with the TabICL preprocessing described below.

Sampling of task partitions. For each generated dataset we draw a random partition $(\mathcal{C}, \mathcal{B}, \mathcal{T})$ with $N \sim \mathrm{Uniform}\{8, \ldots, 1024\}$, buffer capacity fixed at $K = 32$, and target count $M = 512$. Per batch, we fix (d, N, K, M) across tasks to avoid padding and stack samples directly.

Preprocessing. We adopt the TabICL *PreprocessingPipeline* and fit it on context features only. The fitted transform is then applied to context, buffer, and target features. Regression targets are standardized using context statistics, i.e., $\tilde{y} = (y - \mu_{y,\mathcal{C}})/\sigma_{y,\mathcal{C}}$, and the same (μ, σ) are used for buffer and targets. No missing values are synthesized by the SCM generator.

Summary of preprocessing pipeline. We use a three-stage, per-column pipeline following Jingang et al. (2025): (i) standard scaling; (ii) normalization (`power`, i.e., Yeo–Johnson); and (iii) outlier handling via a z-score threshold $\tau = 4.0$. At transform time, values outside the fitted range are clipped to the training (context) min/max before normalization, mirroring TabICL's behavior.

E.1.3 Training procedure

We train with AdamW (learning rate 1×10^{-4}, β=(0.9, 0.95), weight decay 0.0), batch size 64 datasets per step, gradient clipping at 0.5, and dropout 0.0 throughout the backbone. Mixed-precision training uses AMP with `bfloat16`. All runs use `float32` tensors at the data interface. A cosine schedule with warmup is used (`cosine_with_warmup`); `warmup_steps`= 2000 takes precedence over the nominal `warmup_ratio`= 0.20; `num_cycles`= 1. Automatic mixed precision is enabled with `amp_dtype=bfloat16`. Each training step draws a batch of 64 independent tasks (datasets) with feature dimension d sampled from $\{1, \ldots, 10\}$ and context size N from $\{8, \ldots, 1024\}$; buffer size and target count are fixed at K=32 and M=512. Training is capped at `max_steps` = 160,000, i.e., one epoch effective duration. This corresponds to approximately $64 \times 160{,}000 = 10.24$ million synthetic tasks seen during pretraining. The global data seed is 123. We trained the model on a single NVIDIA A100 80 GB GPU for approximately 3 days.

E.2 Evaluation Details

In this paper log-likelihood values are always averaged (LL divided by the number of target points M).

Tabular foundation model. We pretrain a task-agnostic tabular model on synthetic data (Section E.1) and evaluate it on three UCI datasets: Individual Household Electric Power Consumption[1], Gas Turbine CO and NOx Emission DataSet[2], and Bike Sharing[3].

For each dataset, we evaluate likelihood values over 16 randomly sampled subsets. The context and target sets are set to $N = 128, M = 32$. Each likelihood evaluation is an average of 128 permutations.

F Use of Large Language Models

Idea generation and exploration. We used Large Language Models (LLMs) in the early stages of this work to support idea generation, brainstorming, and the exploration of possible methodological directions. LLMs were also employed for tasks such as identifying related work through web search and summarization, which helped us gain an initial overview of relevant literature.

Coding assistant. LLMs provided assistance with coding, primarily by generating boilerplate components of the codebase, visualization scripts, and test codes. They were also used for drafting parts of the implementation in PyTorch. All code produced or suggested by LLMs was carefully reviewed, verified, and modified where necessary to ensure correctness and reliability.

Writing assistant. Finally, LLMs were used in preparing the manuscript, particularly for refining clarity, conciseness, and grammatical correctness. They supported rephrasing and restructuring of text, helping us to communicate ideas more effectively while maintaining the accuracy and integrity of the content.

[1] https://archive.ics.uci.edu/dataset/235/individual+household+electric+power+consumption

[2] https://archive.ics.uci.edu/dataset/551/gas+turbine+co+and+nox+emission+data+set

[3] https://archive.ics.uci.edu/dataset/275/bike+sharing+dataset