ORION-MSP: Multi-Scale Sparse Attention for Tabular In-Context Learning

Mohamed Bouadi
Lexsi Labs
Paris, France
mohamed.bouadi@lexsi.ai

Pratinav Seth
Lexsi Labs
Mumbai, India
pratinav.seth@lexsi.ai

Aditya Tanna
Lexsi Labs
Mumbai, India
aditya.tanna@lexsi.ai

Vinay K. Sankarapu
Lexsi Labs
London, UK
v.k@lexsi.ai

Abstract

Tabular data remain the predominant format for real-world applications. Yet, developing effective neural models for tabular data remains challenging due to heterogeneous feature types and complex interactions occurring at multiple scales. Recent advances in tabular in-context learning (ICL), such as TabPFN and TabICL, have achieved state-of-the-art performance comparable to gradient-boosted trees (GBTs) without task-specific fine-tuning. However, current architectures exhibit key limitations: (1) single-scale feature processing that overlooks hierarchical dependencies, (2) dense attention with quadratic scaling in table width, and (3) strictly sequential processing that prevents iterative refinement and cross-component communication. To address these challenges, we introduce ORION-MSP, a tabular ICL architecture featuring three key innovations: multi-scale processing to capture hierarchical feature interactions, block-sparse attention combining windowed, global, and random patterns for scalable efficiency, and a Perceiver-style memory enabling safe bidirectional information flow across components. Across diverse benchmarks, ORION-MSP matches or surpasses state-of-the-art performance while scaling effectively to high-dimensional tables, establishing a new standard for efficient tabular in-context learning.

1 Introduction

Tabular data remain the most common form of real-world data, spanning critical domains such as healthcare, finance, and scientific research. Despite major advances in deep learning for natural language [17, 28] and vision [5], GBTs remain the state-of-the-art for tabular prediction. Recently, tabular in-context learning (ICL) has emerged as a promising approach that adapts the ICL paradigm—central to large language models—to tabular data. This enables pretraining across diverse tables and rapid task adaptation without gradient updates. TabPFN [9] pioneered this direction by meta-training a transformer on synthetic datasets generated via structural causal models. Its encoder–decoder design lets test samples attend to training examples for zero-shot prediction, but its alternating column- and row-wise attentions make large training sets computationally prohibitive.

A recent variant, TabDPT [18], showed that comparable performance can be achieved on real-world datasets by using similarity-based retrieval to construct contextual examples—an idea first explored in TabR [6]. However, its diffusion process introduces substantial computational overhead, limiting scalability. Building on this, TabPFN-v2 [10] proposed cell-based ICL, extending row-based encoding

AI for Tabular Data workshop at EurIPS 2025.

to datasets exceeding 10,000 samples. This paradigm was further refined by TabICL [21], which designed a table-native transformer with column embedding, row interaction, and ICL prediction modules. While achieving SOTA results across diverse benchmarks, it retained dense attention in row interactions and a strictly sequential pipeline, limiting iterative refinement and cross-module communication. ContextTab [23] later enhanced performance on heterogeneous datasets through contextualized feature embeddings and tabular-specific attention mechanisms.

Collectively, existing tabular ICL models demonstrate strong performance yet share core limitations: dense quadratic attention, uniform single-scale processing, and lack of cross-component feedback.

To overcome these challenges, we propose ORION-MSP, a unified architecture combining multi-scale sparse attention with a Perceiver-style latent memory for efficient and expressive tabular reasoning.

2 Proposed Approach: ORION-MSP

ORION-MSP framework consists of four core components that collectively enable efficient and generalizable tabular in-context learning: (1) Column Embedding: transforms raw tabular features into dense, semantically meaningful representations; (2) Multi-Scale Sparse Row Interaction: captures dependencies at multiple granularities via a hierarchy of attention scales, combining *CLS* and *GLOBAL* tokens for local and long-range connectivity; (3) Cross-Component Perceiver Memory: introduces a latent memory bottleneck that enables safe bidirectional communication between modules, promoting iterative refinement without information leakage; (4) Dataset-wise In-Context Learning Predictor: leverages the enriched representations to perform zero-shot prediction across new tasks without gradient updates. An overview of the complete architecture is shown in Figure 1.

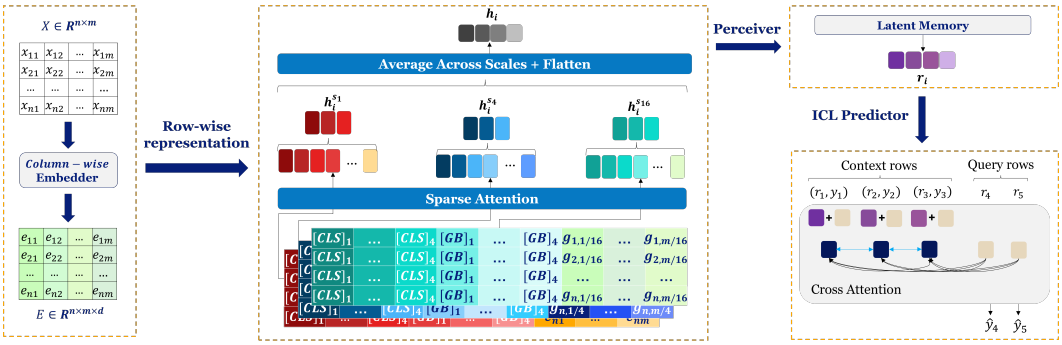


Figure 1: An overview of Orion-MSP architecture. First, column-wise embedding transforms input table into embedding vectors E. Next, multi-scale sparse row interaction prepends learnable *[CLS]* and *[GLOBAL]* tokens to E, processes features at multiple granularities (scales 1, 4, and 16) with sparse attention transformers, and aggregates *[CLS]* outputs across scales to yield row embeddings H. Cross-component Perceiver memory enables bidirectional communication: training rows write to latent memory, which all rows read for enhanced representations R. Finally, ICL predicts test labels from R in a single forward pass.

2.1 Column-wise Embedding

We adopt the original TabICL [21] column-wise embedder to map each cell in a column $c_j \in \mathbb{R}^n$ to a d-dimensional representation using a *shareable Set Transformer*, TF_{col}, that treats the column as a permutation-invariant set of values. Concretely, TF_{col} predicts a per-cell affine map, assigning each cell its own weight and bias: $W, B = TF_{\text{col}}(c_j) \in \mathbb{R}^{n\times d}$, $e_j = W \odot c_j + B \in \mathbb{R}^{n\times d}$.

This frames feature embedding as a set-input hypernetwork problem, producing *distribution-aware* embeddings without feature-specific layers and enabling parameter sharing across columns [21].

2.2 Multi-Scale Feature Processing

Uniform self-attention over all features ignores natural hierarchies common in tabular data. To capture both local and global dependencies at low cost, ORION-MSP processes features at multiple granularities in parallel.

We use powers-of-four granularities ($S = \{1, 4, 16\}$). For a row-level feature set $F = \{f_1, \ldots, f_m\}$, each scale $s \in S$ produces a shorter sequence of groups $G^{(s)} = \{g_1^{(s)}, \ldots, g_{K_s}^{(s)}\}$ with $K_s = \lceil m/s \rceil$. Each $g_i^{(s)}$ is formed using *Pooling by Multihead Attention (PMA)* [15] with K_s learnable queries attending to F, as detailed in Appendix. A.4.

Special tokens and encoding. To enable supervision and cross-row communication, each scale prepends 4 CLS tokens and 4 GLOBAL tokens: $X_s = \big[\mathrm{CLS}_{1:4},\ \mathrm{GLOBAL}_{1:4},\ G^{(s)}\big]$.

A lightweight transformer with efficient (block/sparse) attention encodes $S^{(s)}$ independently per scale. Then scale-specific CLS summaries are merged into a single row representation for downstream ICL: $\mathrm{CLS}_{\text{final}} = \frac{1}{|S|} \sum_{s \in S} \mathrm{CLS}^{(s)}$. We default to mean fusion for simplicity and robustness.

This design preserves fine-grained signals (scale 1) while exposing higher-order patterns, yielding stronger hierarchical feature interaction within a tight parameter budget—see Appendix. A.4

2.3 Block-Sparse Attention Mechanism

Dense self-attention scales quadratically in sequence length, which is prohibitive once multi-scale sequences are introduced. We therefore adopt a block-sparse pattern that preserves local detail, long-range routing, and limited randomness for expressivity—at near-linear cost.

Let N be the sequence length and $\mathcal{A}(i, j)$ the attention mask entry for query i and key j. We compose three complementary sparsity patterns (details and ablations in Appendix. A.4):

(i) Windowed (local) attention. Each token sees a symmetric window of width $w = 8$, so that $\mathcal{A}_{\text{win}}(i, j) = 0$ if $|i - j| \le w$, and $\mathcal{A}_{\text{win}}(i, j) = -\infty$ otherwise.

(ii) Global tokens. The first 8 special tokens (CLS/GLOBAL) are fully connected: $\mathcal{A}_{\text{glob}}(i, j) = 0$ if $i < 8$ or $j < 8$, and $\mathcal{A}_{\text{glob}}(i, j) = \mathcal{A}_{\text{win}}(i, j)$ otherwise.

(iii) Random links. Each non-special token adds $r = 2$ uniformly sampled keys to avoid fragmentation and improve mixing: $\mathcal{A}_{\text{rand}}(i, j) = 0$ if $j \in \mathrm{Rnd}(i)$, $|\mathrm{Rnd}(i)| = r$, and $\mathcal{A}_{\text{rand}}(i, j) = \mathcal{A}_{\text{glob}}(i, j)$ otherwise.

The final mask is $\mathcal{A} = \mathcal{A}_{\text{rand}}$, i.e., windowed local neighborhoods, globally connected special tokens, plus sparse random shortcuts as depicted in Figure 2.

2.4 Cross-Component Communication via Perceiver Memory

A purely sequential pipeline ($TF_{\text{col}} \to TF_{\text{row}} \to TF_{\text{ICL}}$) prevents later components from informing earlier representations. We introduce a lightweight Perceiver-style latent memory that aggregates dataset-level regularities from *training* rows and broadcasts them to all components.

The memory is a set of L learnable slots $\mathbf{M} = \{\mathbf{m}_\ell\}_{\ell=1}^{L}$, $\mathbf{m}_\ell \in \mathbb{R}^d$. We use cross-attention for *write* and *read*: $\underbrace{\mathbf{M}^\star}_{\text{write}} = \mathrm{XAttn}\big(\mathbf{M},\, \mathbf{R}_{\text{train}}\big)$, $\underbrace{\mathbf{R}_{\text{enh}}}_{\text{read}} = \mathrm{XAttn}\big(\mathbf{R}_{\text{all}},\, \mathbf{M}^\star\big)$ where $\mathbf{R}_{\text{train}}$ / $\mathbf{R}_{\text{all}}$ are row representations from training-only / all rows, respectively. (Variants with multi-head cross-attention, residuals, and slot gating are in Appendix. A.5.)

Write uses only training rows; *read* is a deterministic function of $\mathbf{M}^\star$. Hence test rows never influence $\mathbf{M}^\star$ and cannot affect training representations.

2.5 Enhanced In-Context Learning

Following the cross-component communication, we inject labels into the training portion of the enhanced representations: $\mathbf{R}_{enhanced}[:, : n_{train}] \leftarrow \mathbf{R}_{enhanced}[:, : n_{train}] + \textit{Embed}(y_{train})$

The ICL predictor applies split attention to maintain the in-context learning paradigm. This ensures that test tokens can only attend to training tokens, maintaining the causal structure required for ICL.

3 Experimental Evaluation

3.1 Experimental Setup

Our evaluation spans three benchmark suites—TALENT [26] (181 datasets), OPENML-CC18 [2] (72 datasets), and TABZILLA [19] (36 tasks)—providing a broad assessment of tabular learning performance. Experiments were run with the TabTune Library [25], a unified framework that standardizes the full workflow for TFMs. To ensure fairness and reproducibility, we use official dataset splits and report results only for datasets shared across all models. After filtering, the final evaluation includes 154 TALENT, 63 OPENML-CC18, and 27 TABZILLA datasets, with minor exclusions due to OOM or CUDA errors. Higher mean ranks indicate consistency across datasets rather than task-specific peaks in accuracy or F1. Dataset statistics are described in Appendix C.4.

Table 1: Overall performance of tabular classification models on the TALENT, OpenML-CC18, and TabZilla benchmark suites. For each model, we report accuracy-based mean rank (lower is better), accuracy (ACC), and weighted F1 (F1) on each suite, as well as aggregated ranks on all datasets and on the balanced and imbalanced subsets. Formatting: **1st place**; 2nd place.

Models	TALENT			OpenML-CC18			TabZilla			Balanced			Imbalanced		
	Rank	ACC	F1	Rank	ACC	F1	Rank	ACC	F1	Rank	ACC	F1	Rank	ACC	F1
XGBoost	6.02	0.8403	0.8360	5.89	0.8558	0.8537	0.8612	0.8326	–	7.00	0.8175	0.8110	6.23	**0.8859**	**0.8785**
CatBoost	5.57	0.8336	0.8259	6.25	0.8588	0.8520	7.13	0.8579	0.8384	7.15	0.8076	0.8020	5.65	0.8785	0.8665
Random Forest	6.15	0.8285	0.8209	6.36	0.8547	0.8497	8.42	0.8358	0.8399	7.92	0.7983	0.7955	6.77	0.8741	0.8646
LightGBM	6.11	0.8331	0.8245	6.18	0.8581	0.8493	5.25	0.8618	0.8211	7.32	0.8071	0.7977	6.19	0.8775	0.8633
TabICL	4.09	0.8471	0.8379	4.69	0.8667	0.8623	5.89	0.8734	0.8698	4.72	0.8279	0.8233	5.08	0.8806	0.8698
OrionBiX	4.59	0.8346	0.8260	4.98	0.8653	0.8596	4.89	0.8728	0.8628	5.65	0.8096	0.8040	5.04	0.8787	0.8683
OrionMSP	**3.26**	0.8461	0.8360	**4.12**	**0.8722**	**0.8676**	**3.84**	**0.8821**	**0.8786**	4.22	0.8265	0.8202	**3.38**	0.8840	0.8731
TabPFN	3.72	**0.8514**	**0.8412**	4.76	0.8714	0.8663	4.86	0.8752	0.8716	**3.85**	**0.8367**	**0.8309**	5.37	0.8808	0.8697
Mitra	10.38	0.3921	0.2868	10.52	0.3614	0.2522	11.21	0.3152	0.1830	12.26	0.2763	0.1540	11.24	0.4794	0.3858
ContextTab	9.84	0.5474	0.4596	6.28	0.8639	0.8581	7.13	0.8389	0.8334	9.66	0.5079	0.4487	9.72	0.7850	0.7192
TabDPT	5.19	0.8408	0.8318	4.64	0.8672	0.8625	3.94	0.8814	0.8775	5.16	0.8233	0.8189	5.65	0.8798	0.8690

3.2 Results

Table 1 compares all models across TALENT, OPENML-CC18, and TABZILLA, covering both balanced and imbalanced datasets. Classical baselines remain competitive (ACC ≈ 0.83–0.86, mean rank ≈ 6), yet pretrained tabular TFMs consistently surpass them, demonstrating strong zero-shot generalization. ORION-MSP achieves the best overall performance (mean rank 3.58) with ACC/F1 of $0.846/0.836$ on TALENT, $0.872/0.868$ on OPENML-CC18, and $0.882/0.879$ on TABZILLA. TABPFN ranks second (4.61), showing stable cross-benchmark performance.

Partitioning by dataset balance shows ORION-MSP gains most on imbalanced datasets (ACC $=$ 0.884, F1 $= 0.873$), reflecting its ability to model minority classes via multi-scale sparse attention. On balanced data, gains narrow, suggesting its hierarchical design benefits skewed or complex distributions. TABPFN remains robust across both but lags behind on minority-class recognition.

Extended analyses (Appendix C) show that ORION-MSP performs robustly across dataset sizes and feature dimensionalities, excelling on small, narrow datasets while scaling effectively to large, high-dimensional tables. Domain-specific results highlight top performance in finance and strong outcomes in medical tasks, reflecting its ability to model hierarchical and cross-feature dependencies. Overall, ORION-MSP proves especially effective for imbalanced, high-dimensional, and context-rich tabular tasks.

Fine-grained scales capture subtle minority-class signals, while coarser scales aggregate global context, yielding balanced local–global representations. Sparse attention enhances efficiency and regularization, mitigating overfitting in high-dimensional or correlated-feature settings. The Perceiver memory further supports cross-scale reasoning by storing and retrieving non-local patterns—particularly valuable in context-dependent domains. However, its added complexity offers limited benefit on simpler, low-dimensional data, suggesting future work on adaptive scale selection and dynamic sparsity control.

Acknowledgments and Disclosure of Funding

We thank the authors of TabICL [21] for publicly releasing their code, which served as the basis for the first component of our proposed architecture.

References

[1] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *CoRR*, abs/2004.05150, 2020.

[2] Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Pieter Gijsbers, Frank Hutter, Michel Lang, Rafael G Mantovani, Jan N van Rijn, and Joaquin Vanschoren. Openml benchmarking suites. *arXiv preprint arXiv:1708.03731*, 2017.

[3] Stephan Bongers, Patrick Forré, Jonas Peters, and Joris M Mooij. Foundations of structural causal models with cycles and latent variables. *The Annals of Statistics*, 49(5):2885–2915, 2021.

[4] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

[5] Micah Goldblum, Hossein Souri, Renkun Ni, Manli Shu, Viraj Prabhu, Gowthami Somepalli, Prithvijit Chattopadhyay, Mark Ibrahim, Adrien Bardes, Judy Hoffman, et al. Battle of the backbones: A large-scale comparison of pretrained models across computer vision tasks. *Advances in Neural Information Processing Systems*, 36:29343–29371, 2023.

[6] Yury Gorishniy, Ivan Rubachev, Nikolay Kartashev, Daniil Shlenskii, Akim Kotelnikov, and Artem Babenko. Tabr: Tabular deep learning meets nearest neighbors. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[7] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in Neural Information Processing Systems (NeurIPS)*, 35:507–520, 2022.

[8] D Hendrycks. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[9] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[10] 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. *Nat.*, 637(8044):319–326, 2025.

[11] 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*, pages 4651–4664. PMLR, 2021.

[12] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *Advances in neural information processing systems*, 30, 2017.

[14] Alex Krizhevsky, Geoff Hinton, et al. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 40(7):1–9, 2010.

[15] 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*, pages 3744–3753. PMLR, 2019.

[16] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753. PMLR, 2019.

[17] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM computing surveys*, 55(9):1–35, 2023.

[18] Junwei Ma, Valentin Thomas, Rasa Hosseinzadeh, Hamidreza Kamkari, Alex Labach, Jesse C. Cresswell, Keyvan Golestan, Guangwei Yu, Maksims Volkovs, and Anthony L. Caterini. Tabdpt: Scaling tabular foundation models. *CoRR*, abs/2410.18164, 2024.

[19] Duncan McElfresh, Sujay Khandagale, Jonathan Valverde, Ganesh Ramakrishnan, Vishak Prasad, Micah Goldblum, and Colin White. When do neural nets outperform boosted trees on tabular data? In *Advances in Neural Information Processing Systems*, 2023.

[20] Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. In *International Conference on Learning Representations (ICLR)*, 2022.

[21] Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. Tabicl: A tabular foundation model for in-context learning on large data. *CoRR*, abs/2502.05564, 2025.

[22] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

[23] Marco Spinaci, Marek Polewczyk, Maximilian Schambach, and Sam Thelin. Contexttab: A semantics-aware tabular in-context learner. *CoRR*, abs/2506.10707, 2025.

[24] Jianlin Su, Murtadha H. M. Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[25] Aditya Tanna, Pratinav Seth, Mohamed Bouadi, Utsav Avaiya, and Vinay Kumar Sankarapu. Tabtune: A unified library for inference and fine-tuning tabular foundation models. *arXiv preprint arXiv:2511.02802*, 2025.

[26] Han-Jia Ye, Si-Yang Liu, Hao-Run Cai, Qi-Le Zhou, and De-Chuan Zhan. A closer look at deep learning methods on tabular datasets. *arXiv preprint arXiv:2407.00956*, 2024.

[27] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

[28] Fei Yu, Hongbo Zhang, Prayag Tiwari, and Benyou Wang. Natural language reasoning, a survey. *ACM Computing Surveys*, 56(12):1–39, 2024.

[29] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[30] Neil Zeghidour and David Grangier. Wavesplit: End-to-end speech separation by speaker clustering. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:2840–2849, 2021.

A Technical Details - ORION-MSP

A.1 Problem Formulation

Consider a tabular dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ with n samples and m features. Let $\mathbf{X} \in \mathbb{R}^{n \times m}$ denote the feature matrix, where each column $\mathbf{c}_j \in \mathbb{R}^n$ ($j \in \{1, \ldots, m\}$) represents the values of the j-th feature across all samples.

In the in-context learning setting, we are given a *context set* of n_{train} labeled examples:

$$\mathcal{C} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n_{\text{train}}}$$

and a set of n_{test} *query samples*:

$$\mathcal{Q} = \{\mathbf{x}_i\}_{i=1}^{n_{\text{test}}}$$

Our goal is to predict the conditional distribution of the target for each query sample given the context set:

$$p(y|x, \mathcal{C}), \forall x \in \mathcal{Q}$$

A.2 High-level Structure: From Data to ICL

Orion-MSP consists of four core components that collectively enable efficient and generalizable tabular in-context learning: (1) Column Embedding: transforms raw tabular features into dense, semantically meaningful representations; (2) Multi-Scale Sparse Row Interaction: captures dependencies at multiple granularities via an hierarchy of attention scales, combining *CLS* and *GLOBAL* tokens for local and long-range connectivity; (3) Cross-Component Perceiver Memory: introduces a latent memory bottleneck that enables safe bidirectional communication between modules, promoting iterative refinement without information leakage; (4) Dataset-wise In-Context Learning Predictor: leverages the enriched representations to perform zero-shot prediction across new tasks without gradient updates. An overview of the complete architecture is shown in Figure 1.

Orion-MSP extends the original TabICL [21] architecture with three complementary innovations designed to address the fundamental challenges of tabular data processing: computational inefficiency, limited feature interaction modeling, and the need for hierarchical pattern recognition. Our approach maintains the core in-context learning paradigm while introducing architectural enhancements that significantly improve both efficiency and performance.

A.3 Column-wise Embedding

In this subsection, we briefly review the column-wise embedding module originally proposed in TabICL [21], which we adopt as the first component of our architecture.

Tabular data exhibits unique characteristics compared to other modalities: each column represents a distinct feature with its own distribution, scale, and statistical properties (e.g., mean, variance, skewness, kurtosis). To capture these distributional characteristics, we adopt the original TabICL column-wise embedder to map each scalar cell in a column $c_j \in \mathbb{R}^n$ to a d-dimensional representation using a *shareable Set Transformer*, TF_{col}, that treats the column as a permutation-invariant set of values. Our goal is to transform each cell value X_{ij} into a d-dimensional embedding $\mathbf{E}_{ij} \in \mathbb{R}^d$ that encodes both:

1. The value of the cell (X_{ij})
2. The distributional context of the column ($\mathbf{c}_j$)

This differs fundamentally from standard embedding approaches (e.g., word embeddings) where each discrete token has a fixed embedding regardless of context. In tabular data, the *meaning* of a value depends heavily on the column's distribution: a value of 50 may be typical in one feature but an outlier in another.

Concretely, TF_{col} predicts a per-cell affine map, assigning each cell its own weight and bias. The process consists of three main steps:

A.3.1 Initial Projection:

Project the column values into a d-dimensional embedding space:

$$\mathbf{U}_j = \text{Linear}_{\text{proj}}(\mathbf{c}_j) \in \mathbb{R}^{n \times d} \tag{1}$$

where $\text{Linear}_{\text{proj}} : \mathbb{R} \to \mathbb{R}^d$ is a learned linear transformation. This creates initial token embeddings for each cell in the column.

A.3.2 Induced Set Attention Blocks (ISAB):

To efficiently capture global distributional information while maintaining computational tractability, we employ ISAB [16] with k learnable inducing points. It consists of two sequential Multi-Head Attention Blocks $(\text{MAB}_1, \text{MAB}_2)$:

$$\mathbf{M}_j = \text{MAB}_1(\mathbf{I}, \mathbf{U}_j^{\text{train}}, \mathbf{U}_j^{\text{train}}) \in \mathbb{R}^{k \times d} \tag{2}$$

$$\mathbf{V}_j = \text{MAB}_2(\mathbf{U}_j, \mathbf{M}_j, \mathbf{M}_j) \in \mathbb{R}^{n \times d} \tag{3}$$

where $\mathbf{I} \in \mathbb{R}^{k \times d}$ denote k trainable inducing point embeddings ($k \ll n$), which serve as a compressed representation of the column distribution.

We define a Multi-Head Attention Block as:

$$\text{MAB}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{LayerNorm}(\mathbf{H} + \text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V})) \tag{4}$$

where $\mathbf{H}$ is a residual connection (set to $\mathbf{Q}$ if dimensions match, otherwise passed through a projection), and:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}_O \tag{5}$$

with each head defined as:

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_Q^i, \mathbf{K}\mathbf{W}_K^i, \mathbf{V}\mathbf{W}_V^i) \tag{6}$$

Following TabICL [21], we use d=128, k=128, 4 heads, and 3 ISAB blocks. Crucially, in Equation 2, we use only *training samples* $\mathbf{U}_j^{\text{train}} \in \mathbb{R}^{n_{\text{train}} \times d}$ as keys and values. This ensures that the inducing points $\mathbf{M}_j$ capture the distribution of the training data only, preventing information leakage from test samples during embedding. This is crucial for maintaining the in-context learning paradigm.

In Equation 3, all samples (training and test) query the inducing points to obtain their contextualized embeddings. The inducing points act as a distributional summary: they encode statistical properties (e.g., mean, variance, skewness) of the training column values, and each cell embedding is adjusted based on where it lies within this learned distribution.

A.3.3 Weight and Bias Generation:

The ISAB output $\mathbf{V}_j$ is passed through a feedforward network to generate cell-specific weights and biases:

$$\mathbf{W}_j, \mathbf{B}_j = \text{FFN}(\mathbf{V}_j), \mathbf{W}_j, \mathbf{B}_j \in \mathbb{R}^{n \times d} \tag{7}$$

where:

$$\text{FFN}(\mathbf{V}_j) = \text{Linear}_{\text{out}}(\text{GELU}(\text{Linear}_{\text{hidden}}(\mathbf{V}_j))) \tag{8}$$

The final embeddings are then computed as:

$$\mathbf{E}_{:,j,:} = \mathbf{W}_j \odot \mathbf{c}_j + \mathbf{B}_j \in \mathbb{R}^{n \times d} \tag{9}$$

where $\odot$ denotes element-wise (Hadamard) product, and $\mathbf{c}_j$ is broadcasted to shape (n, d).

This formulation allows each cell's embedding to be a *function* of both its raw value ($\mathbf{c}_j$) and the column's learned distributional properties ($\mathbf{W}_j, \mathbf{B}_j$).

Note that, in our architecture, row-wise interaction requires prepending special tokens (e.g., [CLS], [GLOBAL]) to each row. To accommodate these, the column embedding reserves C positions at the beginning of each column:

$$\mathbf{E} \in \mathbb{R}^{n \times (m+C) \times d} \tag{10}$$

For the reserved positions (indices 1 to C), we use a *skippable linear layer* that outputs zeros or small random values:

$$\mathbf{E}_{:,j,:} = \begin{cases} \text{SkipLinear}(\mathbf{c}_j) & \text{if } j \leq C \text{ (reserved)} \\ \mathbf{W}_j \odot \mathbf{c}_j + \mathbf{B}_j & \text{if } j > C \text{ (features)} \end{cases} \tag{11}$$

where SkipLinear is a linear layer with very small initialization, allowing the model to learn appropriate embeddings for reserved positions during training.

The Set Transformer architecture ensures that TF_{col} is *permutation-invariant* with respect to the order of samples within a column. Formally, let $\pi : [T] \to [T]$ be any permutation, and let $\mathbf{c}'_j = \mathbf{P}_\pi \mathbf{c}_j$ where $\mathbf{P}_\pi$ is the corresponding permutation matrix. Then:

$$\text{TF}_{\text{col}}(\mathbf{c}'_j) = \mathbf{P}_\pi \, \text{TF}_{\text{col}}(\mathbf{c}_j) \tag{12}$$

This property is inherited from the attention mechanism in ISAB, where the softmax normalization and weighted aggregation are invariant to input order.

The inducing points $\mathbf{M}_j \in \mathbb{R}^{k \times d}$ learned by the first MAB serve as a *distributional summary* of column j. Empirically, we observe that:

- Columns with similar statistical moments (mean, variance, skewness, kurtosis) have similar inducing point representations (measured by cosine similarity).
- The inducing points capture multi-modal distributions: for categorical features encoded numerically, different modes correspond to different cluster centers in the inducing point space.
- Outliers in $\mathbf{c}_j$ receive distinct embeddings, as their attention weights to $\mathbf{M}_j$ differ significantly from typical values.

A.4 Multi-Scale Sparse Row-Wise Interaction

While column-wise embedding captures distributional properties of individual features, row-wise interaction must model complex dependencies *across features* to extract meaningful sample representations. However, directly applying dense self-attention to all feature tokens incurs quadratic complexity $O(m^2)$ and may overfit when the number of features varies significantly across datasets. To address these challenges, we introduce a **hierarchical multi-scale sparse attention** mechanism that processes features at multiple granularities with efficient block-sparse patterns.

A.4.1 Motivation and Design Principles

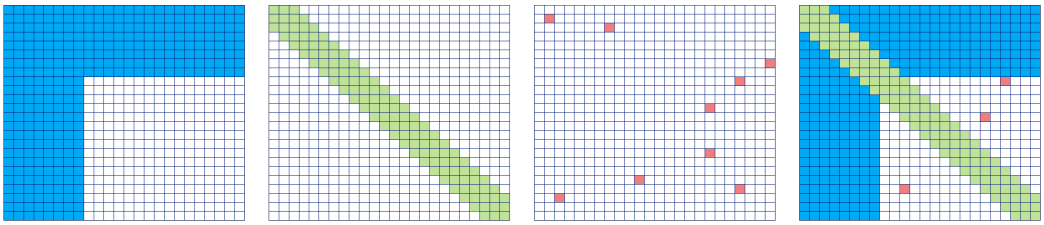
Tabular datasets exhibit several unique characteristics that complicate feature interaction modeling:

1. **Variable feature counts**: The number of features m varies dramatically across datasets, making fixed-scale architectures suboptimal.
2. **Heterogeneous feature relationships**: Some features interact locally (e.g., age and age-related health metrics), while others have global dependencies (e.g., categorical indicators).
3. **Computational constraints**: Dense attention over m features has complexity $O(m^2)$, becoming prohibitive for wide tables or long context windows.

4. **Overfitting risks**: Full attention can memorize training-specific feature correlations that do not generalize to new datasets.

Inspired by hierarchical representations in vision [27] and multi-resolution modeling in speech [30], Orion-MSP decomposes feature interactions into multiple resolution levels:

- **Fine scale** ($s = 1$): Captures detailed pairwise dependencies between individual features.
- **Coarse scales** ($s > 1$): Aggregates semantically related features into groups, reducing sequence length and enabling broader contextual reasoning.
- **Scale aggregation**: Combines representations across scales to balance local precision and global context.



(a) Special Attention (CLS + GLOBAL attention) **(b) Window Attention** **(c) Random Attention** **(d) Orion-MSP Multi-Scale Sparse Attention**

Figure 2: Building blocks of the attention mechanism used in Orion-MSP. White color indicates absence of attention. (a) special attention includes $CLS = 4$ and global attention with $GB = 4$, (b) sliding window attention with $w = 8$, (c) random attention with $r = 2$, (d) the combined row representation of Orion-MSP model.

To further improve efficiency and generalization, we adopt a block-sparse attention pattern inspired by Longformer [1] and BigBird [29], as depicted in Figure 2:

- **Sliding window attention**: Local connectivity within a fixed radius w, preserving fine-grained structure.
- **Global tokens**: Specialized tokens with full connectivity, ensuring stable long-range information flow.
- **Random links**: Optional sparse stochastic connections that enhance expressivity and global reachability..

This design reduces attention complexity from $O(m^2)$ to $O(m \cdot (w + g + r))$ where w, g and r are the window size, the number of global tokens, and the number of random links respectively.

Formally, the multi-scale sparse row-wise transformer, $\text{TF}_{\text{row}}^{\text{MS}}$, processes column-embedded features $\mathbf{E} \in \mathbb{R}^{B \times n \times (m+C) \times d}$ to generate row-wise embeddings $\mathbf{H} \in \mathbb{R}^{B \times n \times (N\text{cls} \cdot d)}$:

$$\mathbf{H} = \text{TF}_{\text{row}}^{\text{MS}}(\mathbf{E}, d_{\text{valid}}) \in \mathbb{R}^{B \times n \times (N_{\text{cls}} \cdot d)} \tag{13}$$

where B is the number of datasets, n the number of samples per dataset, m the number of features, and d the embedding dimension. The constant $C = N_{\text{cls}} + N_{\text{global}}$ accounts for special token slots, and$d_{\text{valid}} \in \mathbb{R}^B$ optionally indicates the number of valid features per dataset for handling variable-length inputs.

The transformation proceeds through the following steps:

A.4.2 Multi-Scale Feature Grouping:

First, for each scale $s \in \mathcal{S} = \{s_1, s_2, \ldots, s_M\}$ (e.g., $\mathcal{S} = \{1, 4, 16\}$), we group the m feature tokens into $K_s = \lceil m/s \rceil$ groups of size s.

The default grouping strategy uses a *learnable soft grouping* via Pooling by Multihead Attention (PMA) [15] to adaptively attend to features:

$$\begin{aligned}
\mathbf{Q}_s &= \text{Seed}_s + \text{PE}(K_s) \in \mathbb{R}^{K_s \times d} \\
\mathbf{K}_s &= \text{Linear}_K(\mathbf{E}_{:,:,C:,:}) \in \mathbb{R}^{B \cdot n \times m \times d} \\
\mathbf{V}_s &= \text{Linear}_V(\mathbf{E}_{:,:,C:,:}) \in \mathbb{R}^{B \cdot n \times m \times d} \\
\mathbf{A}_s &= \text{softmax}\left(\frac{\mathbf{Q}_s \mathbf{K}_s^\top}{\sqrt{d}}\right) \in \mathbb{R}^{K_s \times n} \\
\mathbf{F}_s &= \mathbf{A}_s \mathbf{V}_s \in \mathbb{R}^{B \times n \times K_s \times d}
\end{aligned}$$

where $\text{Seed}_s \in \mathbb{R}^{K_s \times d}$ is a learnable seed embedding, and $\text{PE}(K_s)$ adds sinusoidal positional encodings. PMA allows the model to learn which features to group together, adapting to dataset-specific correlation structures.

A.4.3 Special Tokens Injection:

For each row at each scale, we prepend special tokens:

1. $\text{CLS} \in \mathbb{R}^{N_{\text{cls}} \times d}$ (learnable, per-row summary)
2. $\text{GLOBAL} \in \mathbb{R}^{N_{\text{global}} \times d}$ (learnable, long-range connectivity)

The full sequence at scale s becomes:

$$X_s = [\text{CLS}, \text{GLOBAL}, \text{G}^{(s)}] \in \mathbb{R}^{B \times n \times (N_{\text{special}} + K_s) \times d} \tag{14}$$

where $N_{\text{special}} = N_{\text{cls}} + N_{\text{global}}$.

A.4.4 Block-Sparse Attention Mask:

As depicted in Figure 2, for each scale, we construct a sparse attention mask $\mathbf{M}_s \in \mathbb{R}^{L_s \times L_s}$ where $L_s = N_{\text{special}} + K_s$. The mask follows these rules:

1. **Fully Connected Special Tokens:** The first N_{special} tokens (CLS and GLOBAL) are fully connected to all other tokens and to each other (Figure 2.a):

$$\mathbf{M}_s[i, j] = 0 \quad \forall i \in [1, N_{\text{special}}] \text{ or } j \in [1, N_{\text{special}}] \tag{15}$$

2. **Sliding Window Attention:** Feature tokens (indices $> N_{\text{special}}$) attend to neighbors within a window of radius $w = 8$ (Figure 2.b):

$$\mathbf{M}_s[i, j] = \begin{cases} 0 & \text{if } |i - j| \leq w \text{ and } i, j > N_{\text{special}} \\ -\infty & \text{otherwise} \end{cases} \tag{16}$$

3. **Random Links (Optional):** For each feature token $i > N_{\text{special}}$, we randomly select r additional tokens to attend to (Figure 2.c):

$$\mathbf{M}_s[i, j_k] = 0 \quad \text{for } k \in [1, r],\ j_k \sim \text{Uniform}(\{N_{\text{special}} + 1, \ldots, L_s\} \setminus \{i\}) \tag{17}$$

The final mask is (Figure 2.d):

$$\mathbf{M}_s[i, i] = 0 \quad \forall i \in [1, L_s] \quad \text{(self-attention always allowed)} \tag{18}$$

A.4.5 Transformer Encoder per Scale:

For each scale $s \in \mathcal{S}$, we apply a dedicated Transformer encoder:

$$\mathbf{Z}_s = \text{Encoder}_s(\mathbf{X}_s, \mathbf{M}_s) \in \mathbb{R}^{B \times n \times L_s \times d} \tag{19}$$

where Encoder_s consists of $N_{\text{blocks}}^{\text{row}} / |\mathcal{S}|$ stacked Transformer blocks with:

- Multi-head self-attention: $\text{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{M}_s)$ with sparse mask $\mathbf{M}_s$
- Rotary positional encoding (RoPE): Applied to queries and keys before attention [24]
- Feed-forward network: Two-layer MLP with GELU activation
- Pre-norm architecture: Layer normalization before each sub-layer

The multi-head attention with sparse masking is computed as:

$$\text{head}_i = \text{Attention}(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i, \mathbf{M}_s) \tag{20}$$

$$= \text{softmax}\left(\frac{\mathbf{Q}_i \mathbf{K}_i^\top}{\sqrt{d_k}} + \mathbf{M}_s\right)\mathbf{V}_i \tag{21}$$

where $\mathbf{M}_s$ contains 0 for allowed positions and $-\infty$ for disallowed positions (additive masking).

After processing through Encoder_s, we extract the CLS token representations:

$$\mathbf{H}_s = \mathbf{Z}_s[:, :, 1 : N_{\text{cls}}, :] \in \mathbb{R}^{B \times n \times N_{\text{cls}} \times d} \tag{22}$$

These represent the row-wise features at scale s. We then aggregate these representations across all scales by averaging:

$$\mathbf{H}_{\text{agg}} = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \mathbf{H}_s \in \mathbb{R}^{B \times n \times N_{\text{cls}} \times d} \tag{23}$$

This simple averaging strategy ensures that each scale contributes equally, balancing fine-grained and coarse-grained information. Next, the CLS tokens are flattened and normalized to produce the final row embeddings:

$$\mathbf{H} = \text{LayerNorm}(\text{Flatten}(\mathbf{H}_{\text{agg}})) \in \mathbb{R}^{B \times n \times (N_{\text{cls}} \cdot d)} \tag{24}$$

where Flatten concatenates the N_{cls} token embeddings.

Algorithm 1 summarizes the complete multi-scale sparse row-wise interaction process.

A.4.6 Computational Complexity

For a given scale s with $K_s = \lceil m/s \rceil$ grouped feature tokens, the per-layer computational complexity of the sparse attention mechanism is:

$$O(B \cdot n \cdot L_s \cdot (w + N_{\text{global}} + r) \cdot d) \tag{25}$$

where B is the batch size, n the number of samples per dataset, m the number of features, d the embedding dimension, and w, N_{global}, and r denote the sliding-window size, number of global tokens, and number of random links, respectively.

For M scales and a total of $N_{\text{blocks}}^{\text{row}}$ Transformer layers distributed evenly across scales, the overall complexity becomes:

$$O_{\text{total}} = \sum_{s \in \mathcal{S}} O(B \cdot n \cdot K_s \cdot (w + N_{\text{global}} + r) \cdot d \cdot \frac{N_{\text{blocks}}^{\text{row}}}{M}) \tag{26}$$

Since $\sum_{s \in \mathcal{S}} K_s \approx m \cdot \left(1 + \frac{1}{s_2} + \ldots + \frac{1}{s_M}\right)$ and typically $w, N_{\text{global}}, r \ll m$, this simplifies to:

$$O_{\text{total}} \approx O(B \cdot n \cdot m \cdot (w + N_{\text{global}} + r) \cdot d \cdot N_{\text{blocks}}^{\text{row}}) \tag{27}$$

compared to the dense attention cost of $O(B \cdot n \cdot m^2 \cdot d \cdot N_{\text{blocks}}^{\text{row}})$.

For typical hyperparameters ($m \in [10, 100]$, $w = 8$, $N_{\text{global}} = 4$, $r = 2$), this results in a reduction from quadratic $O(m^2)$ to near-linear $O(m \cdot 14)$ complexity—achieving linear scaling while preserving both local and global feature dependencies.

Algorithm 1 Multi-Scale Sparse Row-Wise Interaction ($\text{TF}_{\text{row}}^{\text{MS}}$)

Require: Embeddings $\mathbf{E} \in \mathbb{R}^{B \times n \times (m+C) \times d}$, valid features $d_{\text{valid}} \in \mathbb{R}^{B}$
Require: Scales $\mathcal{S} = \{s_1, s_2, \ldots, s_M\}$, window w, random links r
Ensure: Row embeddings $\mathbf{H} \in \mathbb{R}^{n \times m \times (N_{\text{cls}} \cdot d)}$
Initialize learnable tokens $\mathbf{CLS} \in \mathbb{R}^{N_{\text{cls}} \times d}$, $\mathbf{GLOBAL} \in \mathbb{R}^{N_{\text{global}} \times d}$
$\mathbf{H}_{\text{all}} \leftarrow []$ {Store CLS outputs from all scales}
for each scale $s \in \mathcal{S}$ **do**
  $K_s \leftarrow \lceil m/s \rceil$ {Number of groups at scale s}
  // Feature Grouping
  $\mathbf{G}^{(s)} \leftarrow \text{PMA}(\mathbf{E}[:, :, C:, :], K_s)$
  // Construct Sequence
  $\mathbf{X}_s \leftarrow [\mathbf{CLS}, \mathbf{GLOBAL}, \mathbf{G}^{(s)}]$ {Shape: (B, n, L_s, d) where $L_s = N_{\text{special}} + K_s$}
  // Build Sparse Mask
  $\mathbf{M}_s \leftarrow \text{BuildBlockSparseMask}(L_s, N_{\text{special}}, w, r)$
  // Process Through Transformer
  $\mathbf{Z}_s \leftarrow \text{Encoder}_s(\mathbf{X}_s, \mathbf{M}_s)$ {Transformer with RoPE and sparse attention}
  // Extract CLS Tokens
  $\mathbf{H}_s \leftarrow \mathbf{Z}_s[:, :, 1 : N_{\text{cls}}, :]$
  $\mathbf{H}_{\text{all}}.\text{append}(\mathbf{H}_s)$
end for
// Aggregate Across Scales
$\mathbf{H}_{\text{agg}} \leftarrow \frac{1}{|\mathcal{S}|} \sum_{s=1}^{|\mathcal{S}|} \mathbf{H}_{\text{all}}[s]$
// Flatten and Normalize
$\mathbf{H} \leftarrow \text{LayerNorm}(\text{Flatten}(\mathbf{H}_{\text{agg}}))$
return $\mathbf{H}$

A.5 Cross-Component Memory with Perceiver Architecture

While the column-wise embedding and row-wise interaction components of tabular transformers independently model feature- and sample-level dependencies, richer contextual understanding can emerge if information is shared across these components. However, direct cross-component communication poses a major risk to the in-context learning (ICL) paradigm: naive attention between components can leak test-set information, violating the principle that predictions for test samples must depend solely on training examples and the test input itself.

To overcome this limitation, we introduce a Perceiver-style latent memory module [11] that enables safe, leak-free communication between architectural components. This latent memory acts as a shared representation space that can be written to by training samples and read from by both training and test samples, ensuring compliance with ICL constraints while promoting global knowledge sharing.

In standard transformer-based tabular architectures such as TabICL [21], model components operate in a strictly sequential and isolated fashion:

1. Column Embedding (TF_{col}): Encodes feature-wise statistics across samples to capture column-level distributions.
2. Row Interaction (TF_{row}): Models dependencies across features within each sample.
3. ICL Prediction (TF_{icl}): Performs in-context learning to infer test labels from training examples.

This separation simplifies optimization and ensures ICL safety, but also introduces significant limitations:

- No backward adaptation: Column embeddings cannot adjust based on row-level feature interactions.
- Limited contextual refinement: Row-level interactions lack access to global, dataset-level statistics beyond static column embeddings.

- Dataset isolation: Each dataset is processed independently, preventing cross-dataset generalization within a batch.

A fundamental ICL constraint is that test samples must not influence the model's internal state in a way that affects training representations. Formally, letting

$$\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n_{\text{train}}}, \mathcal{D}_{\text{test}} = \{\mathbf{x}_j\}_{j=n_{\text{train}}+1}^{n} \tag{28}$$

the prediction for a test sample $\mathbf{x}_j$ must satisfy:

$$\mathbb{P}(\hat{y}_j \mid \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}) = \mathbb{P}(\hat{y}_j \mid \mathcal{D}_{\text{train}}, \mathbf{x}_j) \tag{29}$$

That is, the prediction depends only on the training set and the test features, never on other test representations or their labels.

Inspired by the Perceiver architecture [11], we introduce a learnable latent memory $\mathbf{L} \in \mathbb{R}^{P \times d}$ with P memory slots. The key idea is:

1. Write Phase (train-only): Memory attends to training representations to extract relevant global patterns.
2. Read Phase (all samples): Both training and test samples attend to the memory to retrieve learned context, but cannot modify it.

This asymmetry guarantees ICL safety, since only training data influence the memory's contents. The memory serves as a compressed, permutation-invariant summary of the training context that enables consistent feature refinement across samples.

The memory module is incorporated inside the ICL transformer (TF_{icl}), refining the row embeddings before label injection and prediction. Given row embeddings $\mathbf{H} \in \mathbb{R}^{B \times n \times d_h}$ (where B is the batch size and n the number of samples per dataset), the Perceiver memory transformation produces refined representations:

$$\mathbf{R} = \text{PerceiverMemory}(\mathbf{H}, n_{\text{train}}) \in \mathbb{R}^{B \times n \times d_H} \tag{30}$$

with:

- $d_H = N_{\text{cls}} \cdot d$ the hidden dimension after multi-head projection,
- P the number of latent memory slots (a hyperparameter),
- and n_{train} the number of labeled training examples.

The Perceiver memory consists of three key stages, each composed of multiple cross-attention layers with residual connections and feed-forward transformations.

1. **Latent Memory Initialization:** We initialize a set of P learnable latent vectors:

   $$\mathbf{L}_0 \in \mathbb{H}^{P \times d_H} \tag{31}$$

   drawn from a truncated normal distribution $\mathcal{N}(0, 0.02^2)$. These latents act as a universal memory bank, shared across all datasets in the batch and reused across forward passes, providing a stable foundation for information aggregation.

2. **Cross-Attention Block:** At the core of the memory is a cross-attention mechanism allowing one representation to attend to another. Given query set $\mathbf{Q}$ and key–value set $\mathbf{KV}$, we define:

   $$\text{CrossAttn}(\mathbf{Q}, \mathbf{KV}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{W}_Q(\mathbf{K}\mathbf{V}\mathbf{W}_K)^\top}{\sqrt{d_k}}\right)(\mathbf{K}\mathbf{V}\mathbf{W}_V) \tag{32}$$

   where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d_R \times d_k}$ are projection matrices and $d_k = d_R/h$ is the per-head dimension.

Each cross-attention block is followed by layer normalization, residual connections, and a feed-forward layer:

$$\mathbf{Q}' = \text{LayerNorm}(\mathbf{Q}) \tag{33}$$

$$\mathbf{KV}' = \text{LayerNorm}(\mathbf{KV}) \tag{34}$$

$$\mathbf{Z} = \mathbf{Q} + \text{MultiHeadCrossAttn}(\mathbf{Q}', \mathbf{KV}') \tag{35}$$

$$\mathbf{Z}' = \mathbf{Z} + \text{FFN}(\text{LayerNorm}(\mathbf{Z})). \tag{36}$$

This block structure ensures stable training and supports multi-head feature integration.

3. **Write Phase: Memory Encoding:** In the write phase, the memory attends to *training samples only* to extract and store relevant patterns. For each dataset b in the batch:

$$\mathbf{H}^{(b)}_{\text{train}} = \mathbf{H}^{(b)}[: T_{\text{train}}, :] \in \mathbb{R}^{n_{\text{train}} \times d_H} \tag{37}$$

and initialize the dataset-specific memory as $\mathbf{L}^{(b)}_0 = \mathbf{L}_0$.
We then apply N_{write} cross-attention blocks where memory latents query the training representations:

$$\mathbf{L}^{(b)}_{i+1} = \text{CrossAttnBlock}(\mathbf{Q} = \mathbf{L}^{(b)}_i, \mathbf{KV} = \mathbf{H}^{(b)}_{\text{train}}) \quad \text{for } i = 0, \ldots, N_{\text{write}} - 1 \tag{38}$$

The final encoded memory is:

$$\mathbf{L}^{(b)} = \mathbf{L}^{(b)}_{N_{\text{write}}} \in \mathbb{R}^{P \times d_H} \tag{39}$$

Importantly, $\mathbf{L}^{(b)}$ depends only on training representations, ensuring no test leakage.

4. **Read Phase: Sample Refinement:** In the read phase, *all samples* (training and test) attend to the memory to retrieve stored context. For dataset b:

$$\mathbf{H}^{(b)}_0 = \mathbf{H}^{(b)} \in \mathbb{R}^{n \times d_H} \tag{40}$$

We apply N_{read} cross-attention blocks where sample queries attend to the memory:

$$\mathbf{R}^{(b)}_{i+1} = \text{CrossAttnBlock}(\mathbf{Q} = \mathbf{R}^{(b)}_i, \mathbf{KV} = \mathbf{L}^{(b)}) \quad \text{for } i = 0, \ldots, N_{\text{read}} - 1 \tag{41}$$

The final refined embeddings are:

$$\mathbf{R}^{(b)} = \mathbf{R}^{(b)}_{N_{\text{read}}} \in \mathbb{R}^{n \times d_R} \tag{42}$$

This asymmetric read–write design preserves the integrity of in-context learning:

- Only training samples write to the memory.
- Both training and test samples read from it.
- The memory functions as a shared, compressed abstraction of the training data that can be safely leveraged for inference.

The complete ICL forward pass with Perceiver memory is described in Algorithm 2:

A.6 Dataset-wise In-Context Learning

After column-wise embedding, multi-scale sparse row-wise interaction, and optional cross-component memory refinement, each sample is represented by a fixed-dimensional row embedding:

$$\mathbf{R} \in \mathbb{R}^{B \times n \times d_R} \tag{43}$$

Algorithm 2 ICL with Perceiver Memory

Require: Row embeddings $\mathbf{H} \in \mathbb{R}^{B \times n \times d_H}$, training labels $\mathbf{y}_{\text{train}} \in \mathbb{R}^{B \times n_{\text{train}}}$
Ensure: Predictions $\hat{\mathbf{y}} \in \mathbb{R}^{B \times (n - n_{\text{train}}) \times C}$ for C classes

// Perceiver Memory (optional)
if $P > 0$ **then**
  for each dataset $b = 1$ to B **do**
    $\mathbf{H}^{(b)}_{\text{train}} \leftarrow \mathbf{H}^{(b)}[: T_{\text{train}}, :]$ {Extract training samples}
    $\mathbf{L}^{(b)} \leftarrow \mathbf{L}_0$ {Initialize memory}
    // Write: Memory attends to training samples
    for $i = 1$ to N_{write} **do**
      $\mathbf{L}^{(b)} \leftarrow \text{CrossAttnBlock}(\mathbf{L}^{(b)}, \mathbf{H}^{(b)}_{\text{train}})$
    end for
    // Read: All samples attend to memory
    $\mathbf{R}^{(b)} \leftarrow \mathbf{H}^{(b)}$
    for $i = 1$ to N_{read} **do**
      $\mathbf{R}^{(b)} \leftarrow \text{CrossAttnBlock}(\mathbf{R}^{(b)}, \mathbf{L}^{(b)})$
    end for
  end for
  $\mathbf{R} \leftarrow \mathbf{R}$ {Use refined embeddings}
end if
// Label Injection (training samples only)
$\mathbf{R}[:, : n_{\text{train}}, :] \leftarrow \mathbf{H}[:, : n_{\text{train}}, :] + \text{OneHot}(\mathbf{y}_{\text{train}})\mathbf{W}_{\text{label}}$
// ICL Transformer with Split Mask
$\mathbf{H} \leftarrow \text{TF}_{\text{icl}}(\mathbf{H}, \text{attn\_mask} = n_{\text{train}})$ {Prevent test-to-train leakage}
// Prediction Head
$\mathbf{R} \leftarrow \text{LayerNorm}(\mathbf{R})$
$\text{logits} \leftarrow \text{FFN}_{\text{decoder}}(\mathbf{R}[:, n_{\text{train}} :, :])$ {Predict test labels only}
return logits

where B is the number of datasets in the batch, n the total number of samples per dataset, and d_R the embedding dimension.

The final component, dataset-wise in-context learning (TF_{icl}), leverages these embeddings to predict test labels by conditioning on labeled training examples—all within a single forward pass and without any gradient-based parameter updates..

Formally, for each dataset b in the batch:

$$\mathcal{D}^{(b)}_{\text{train}} = \{(\mathbf{R}^{(b)}_i, y^{(b)}_i)\}_{i=1}^{n_{\text{train}}} \tag{44}$$

$$\mathcal{D}^{(b)}_{\text{test}} = \{\mathbf{R}^{(b)}_j\}_{j=n_{\text{train}}+1}^{n} \tag{45}$$

The objective is to predict test labels $\hat{y}^{(b)}_j$ for $j > n_{\text{train}}$ using in-context reasoning from training examples only.

$$\hat{\mathbf{y}}_{\text{test}} = \text{TF}_{\text{icl}}(\mathbf{R}, \mathbf{y}_{\text{train}}) \tag{46}$$

The ICL module consists of three main stages:

1. **Label Encoding and Injection:** To ensure consistency across datasets with potentially different label spaces, training labels $\mathbf{y}_{\text{train}} \in \mathbb{R}^{B \times n_{\text{train}}}$ are first normalized to contiguous indices:

$$\tilde{y}_i = \text{argsort}(\text{unique}(\mathbf{y}_{\text{train}}))[\mathbf{y}_{\text{train}}[i]] \tag{47}$$

   mapping any label set $\{2, 5, 9\} \to \{0, 1, 2\}$.

Normalized labels are embedded using one-hot encoding followed by a linear projection:

$$\mathbf{e}_y = \text{OneHot}(\tilde{y}, C_{\text{max}}) \cdot \mathbf{W}_y \in \mathbb{R}^{d_R} \tag{48}$$

where C_{max} is the maximum number of classes (e.g., $C_{\text{max}} = 10$), and $\mathbf{W}_y \in \mathbb{R}^{C_{\text{max}} \times d_R}$ is a learned projection matrix.

Label embeddings are injected *only* into training samples via additive combination:

$$\mathbf{R}[:, : n_{\text{train}}, :] \leftarrow \mathbf{R}[:, : n_{\text{train}}, :] + \mathbf{e}_y(\mathbf{y}_{\text{train}}) \tag{49}$$

ensuring test samples remain unaffected and ICL constraints are preserved.

2. **Split-Masked Transformer:** The augmented embeddings R are processed by a split-masked Transformer, enforcing ICL-safe attention between training and test samples. The attention mask $\mathbf{M}_{\text{split}}$ is defined as:

$$\mathbf{M}_{\text{split}}[i, j] = \begin{cases} 0 & \text{if } i \leq n_{\text{train}} \text{ and } j \leq n_{\text{train}} \quad \text{(train-to-train)} \\ 0 & \text{if } i > n_{\text{train}} \text{ and } j \leq n_{\text{train}} \quad \text{(test-to-train)} \\ -\infty & \text{if } i \leq n_{\text{train}} \text{ and } j > n_{\text{train}} \quad \text{(train-to-test: blocked)} \\ 0 & \text{if } i > n_{\text{train}} \text{ and } j > n_{\text{train}} \quad \text{(test-to-test)} \end{cases} \tag{50}$$

No leakage from test to train samples.

- Training samples attend only to other training samples (learn from labeled context).
- Test samples attend to training samples and other test samples (contextual reasoning).
- No leakage from test to train samples.

The Transformer applies N_{icl} blocks of multi-head self-attention and feed-forward layers:

$$\mathbf{H}^{(0)} = \mathbf{R} \tag{51}$$

$$\mathbf{H}^{(\ell+1)} = \text{TransformerBlock}(\mathbf{H}^{(\ell)}, \mathbf{M}_{\text{split}}) \quad \text{for } \ell = 0, \ldots, N_{\text{icl}} - 1 \tag{52}$$

with the final output normalized via:

$$\mathbf{H} = \text{LayerNorm}(\mathbf{H}^{(N_{\text{icl}})}) \tag{53}$$

3. **Prediction head:** Test sample representations $\mathbf{H}[:, n_{\text{train}} :, :]$ are passed through a two-layer MLP decoder:

$$\mathbf{z} = \text{GELU}(\mathbf{H}[:, n_{\text{train}} :, :]\mathbf{W}_1 + \mathbf{b}_1) \in \mathbb{R}^{B \times n_{\text{test}} \times 2d_R} \tag{54}$$

$$\text{logits} = \mathbf{z}\mathbf{W}_2 + \mathbf{b}_2 \in \mathbb{R}^{B \times n_{\text{test}} \times C_{\text{max}}} \tag{55}$$

Predictions are obtained via softmax with temperature τ:

$$\hat{\mathbf{y}}_{\text{test}} = \text{softmax}(\text{logits}[:, :, : K]/\tau) \tag{56}$$

where K is the number of classes in the current dataset (inferred from training labels), and $\tau = 0.9$ by default.

When the number of classes $K > C_{\text{max}}$ (e.g., $K > 10$), we employ a **hierarchical classification** strategy: Grouping: Partition

(a) Grouping: Partition K classes into $G = \lceil K/C_{\text{max}} \rceil$ balanced groups.
(b) First-level prediction: Predict which group a test sample belongs to.
(c) Second-level prediction: For each group, train a classifier on the subset of classes within that group.
(d) Combination: Multiply group probability with intra-group probability to obtain final prediction.

This hierarchical mechanism preserves the ICL paradigm while scaling to hundreds of classes.

Algorithm 3 Dataset-wise In-Context Learning

Require: Row embeddings $\mathbf{R} \in \mathbb{R}^{B \times n \times d_R}$, training labels $\mathbf{y}_{\text{train}} \in \mathbb{R}^{B \times n_{\text{train}}}$
Ensure: Predictions $\hat{\mathbf{y}}_{\text{test}} \in \mathbb{R}^{B \times (n - n_{\text{train}}) \times K}$
// Optional: Perceiver Memory
if memory enabled **then**
  $\mathbf{R} \leftarrow \text{PerceiverMemory}(\mathbf{R}, n_{\text{train}})$
end if
// Label Encoding and Injection
$\tilde{\mathbf{y}}_{\text{train}} \leftarrow \text{NormalizeLabels}(\mathbf{y}_{\text{train}})$ {Map to $\{0, 1, \ldots, K-1\}$}
$\mathbf{e}_y \leftarrow \text{OneHotLinear}(\tilde{\mathbf{y}}_{\text{train}})$ {Shape: $(B, n_{\text{train}}, d_R)$}
$\mathbf{R}[:, : n_{\text{train}}, :] \leftarrow \mathbf{R}[:, : n_{\text{train}}, :] + \mathbf{e}_y$
// Split-Masked Transformer
$\mathbf{M}_{\text{split}} \leftarrow \text{BuildSplitMask}(n, n_{\text{train}})$
$\mathbf{H} \leftarrow \text{TF}_{\text{icl}}(\mathbf{R}, \mathbf{M}_{\text{split}})$
$\mathbf{H} \leftarrow \text{LayerNorm}(\mathbf{H})$
// Prediction Head
$\mathbf{H}_{\text{test}} \leftarrow \mathbf{H}[:, n_{\text{train}} :, :]$ {Extract test representations}
$\text{logits} \leftarrow \text{MLP}_{\text{decoder}}(\mathbf{H}_{\text{test}})$ {Shape: $(B, T_{\text{test}}, C_{\max})$}
$\text{logits} \leftarrow \text{logits}[:, :, : K]$ {Select active classes}
$\hat{\mathbf{y}}_{\text{test}} \leftarrow \text{softmax}(\text{logits}/\tau)$
return $\hat{\mathbf{y}}_{\text{test}}$

During pretraining, the model is trained with cross-entropy loss on test samples:

$$\mathcal{L} = -\frac{1}{B \cdot n_{\text{test}}} \sum_{b=1}^{B} \sum_{j=n_{\text{train}}+1}^{n} \log p(y_j^{(b)} \mid \mathbf{R}^{(b)}, \mathbf{y}_{\text{train}}^{(b)}) \tag{57}$$

Critically, gradients flow through the entire architecture (column embedding, row interaction, memory, ICL transformer, decoder) in an end-to-end manner, enabling the model to learn representations optimized for in-context learning.

B Pretraining and Implementation Details

B.1 Pretraining Data Generation

Following the pretraining paradigm of TabICL [21], we train our model on synthetically generated datasets to learn generalizable representations for in-context learning on tabular data. Unlike natural language or vision domains where large-scale real data is available, tabular datasets exhibit extreme heterogeneity in schemas, distributions, and task objectives. Synthetic data generation via structural causal models (SCMs) enables us to control dataset diversity while ensuring coverage of diverse statistical patterns.

B.1.1 Structural Causal Model (SCM) Prior

We generate synthetic datasets using SCM-based priors [3, 20], where features are related through nonlinear causal relationships. For a dataset with m features, we define a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where each node $v \in \mathcal{V}$ represents a feature, and edges $\mathcal{E}$ encode causal dependencies.

Each feature X_j is computed as:

$$X_j = f_j(\text{Pa}(X_j), \epsilon_j) \tag{58}$$

where $\text{Pa}(X_j)$ are the parent features of X_j in $\mathcal{G}$, f_j is a nonlinear activation function, and $\epsilon_j \sim \mathcal{N}(0, \sigma_j^2)$ is Gaussian noise.

Activation Function Diversity To ensure broad coverage of feature transformations observed in real-world tabular data, we used the following activation functions:

- **Identity**,
- $\tanh$,
- **LeakyReLU**,
- **ELU**
- **Standard nonlinearities**: ReLU, ReLU6 [14], SELU [13], SiLU (Swish) [8], Softplus
- **Bounded functions**: Hardtanh$(x) = \max(-1, \min(1, x))$, Signum function sgn$(x)$
- **Periodic functions**: $\sin(x)$ (captures cyclic patterns, e.g., time-of-day)
- **Radial basis function**: RBF$(x) = \exp(-x^2)$ (models local interactions)
- **Exponential growth/decay**: $\exp(x)$ (models compounding effects, e.g., financial data)
- **Power functions**: $f(x) = \sqrt{|x|}$, $f(x) = x^2$, $f(x) = |x|$ (models scaling relationships)
- **Indicator function**: $f(x) = \mathbb{1}_{|x| \leq 1}$ (models threshold effects)
- **Random Fourier features**: $f(x) = \boldsymbol{\phi}(x)^\top \mathbf{z}$ where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and the feature map $\boldsymbol{\phi} : \mathbb{R} \to \mathbb{R}^N$ is defined as:

$$\phi_i(x) := \frac{w_i}{\|\mathbf{w}\|_2} \cdot \sin(a_i x + b_i), \quad i \in \{1, \ldots, N\} \tag{59}$$

  with $N = 256$, $b_i \sim \mathcal{U}[0, 2\pi]$, $a_i \sim \mathcal{U}[0, N]$, and $w_i := a_i^{-\exp(u)}$ where $u \sim \mathcal{U}[0.7, 3.0]$. This random function approximates complex, non-parametric relationships [22].

For each feature X_j, an activation function f_j is sampled uniformly from this extended set, ensuring diverse nonlinear transformations across the dataset.

B.1.2 Tree-Based SCM Prior

To complement the continuous SCM prior, we also generate datasets using a tree-based SCM prior [7], where the causal mechanism f_j is a decision tree or random forest. This prior is particularly important for modeling categorical interactions and hierarchical decision boundaries commonly observed in real-world tabular data (e.g., credit scoring, medical diagnosis).

For each feature X_j, we construct a random decision tree $\mathcal{T}_j$ with:

- **Splitting criteria**: Random thresholds on parent features Pa(X_j)
- **Leaf values**: Sampled from a Gaussian or uniform distribution
- **Tree depth**: Sampled uniformly from $\{1, 2, 3, 4\}$ to vary complexity

The tree-based prior generates datasets with *piecewise-constant* relationships, contrasting with the smooth transformations of the MLP-based SCM prior.

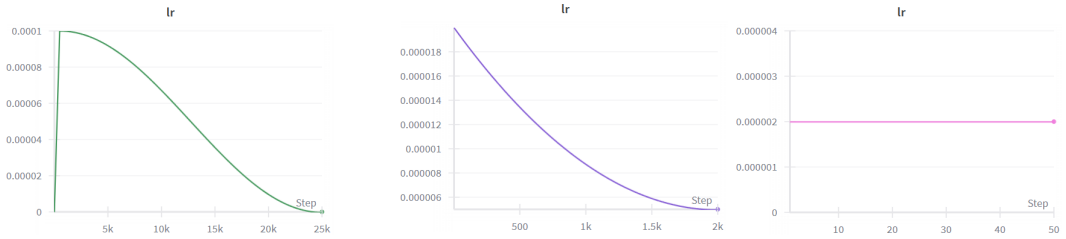
B.2 Pretraining Details

We employ a three-stage curriculum learning strategy that progressively increases dataset size (number of samples per dataset) and refines different architectural components.

1. **Stage 1** (25K steps, 2,048 datasets) trains all components end-to-end with $N_B = 8$ micro-batches for gradient accumulation, where each dataset contains a fixed size of 1,024 samples. This stage establishes foundational representations across column embedding, multi-scale sparse row interaction, Perceiver memory, and ICL prediction.
2. **Stage 2** (2K steps, 512 datasets) reduces micro-batch size to $N_B = 1$ and samples dataset sizes from a log-uniform distribution $\mathcal{U}_{\log}[1024, 40000]$, exposing the model to variable context lengths while maintaining architectural diversity. Within each micro-batch, all datasets share the same sample count, but this count varies across micro-batches.

3. **Stage 3** (50 steps, 512 datasets) focuses exclusively on long-context ICL by freezing all components except TF_{icl} and sampling dataset sizes uniformly from $\mathcal{U}[40000, 60000]$, ensuring robust in-context learning on large datasets.

Across all stages, we use the Adam optimizer [12] with gradient norm clipping at 1.0 and a learning rate schedule shown in Figure 3. This curriculum, progressing from small, uniform datasets to large, variable datasets with selective fine-tuning, enables the model to generalize effectively across diverse dataset scales while preventing overfitting to specific sample counts.



(a) Cosine decay for stage 1 (b) Polynomial decay for stage 2 (c) Constant lr for stage 3

Figure 3: Learning rate schedules for pretraining stages.

B.3 Implementation Details

This section provides comprehensive hyperparameters and training configurations for all architectural components of ORION-MSP. Our implementation is built on PyTorch and trained on NVIDIA H200 GPUs.

B.3.1 Column-wise Embedding (TF_{col})

Hyperparameters

- Embedding dimension: $d = 128$
- Number of inducing points: $k = 128$
- Attention heads: $h = 4$ (head dimension $d_k = 32$)
- ISAB blocks: $L = 3$
- Feedforward dimension: $d_{\text{ff}} = 256$
- Dropout rate: $p = 0.0$
- Activation: GELU
- Layer norm: Pre-norm

Initialization and Training Considerations

- Inducing points initialized from $\mathcal{N}(0, 0.02^2)$
- Linear and attention weights: Xavier/Glorot uniform
- Column dropout: $p_{\text{col}} = 0.1$
- Gradient clipping: max-norm = 1.0

B.3.2 Multi-Scale Sparse Row Interaction ($\text{TF}_{\text{row}}^{\text{MS}}$)

Hyperparameters

- Embedding dimension: $d = 128$
- Transformer blocks: $N_{\text{blocks}}^{\text{row}} = 6$ (2 per scale)
- Scales: $\mathcal{S} = \{1, 4, 16\}$

- Attention heads: $h = 8$
- Window size: $w = 8$, Global tokens: $N_{\text{global}} = 4$, Random links: $r = 2$
- Dropout: $p = 0.0$
- Positional encoding: RoPE ($\theta = 100000$)

Training Details

- Sparse attention via PyTorch SDPA
- Cosine learning rate schedule with 5% warmup
- Mixed precision: FP16 (forward), FP32 (softmax)

B.3.3 Cross-Component Perceiver Memory

Hyperparameters

- Memory slots: $P = 32$
- Write layers: $N_{\text{write}} = 2$, Read layers: $N_{\text{read}} = 2$
- Attention heads: $h = 4$
- Feedforward dimension: $d_{\text{ff}} = 2d_R$
- Dropout: $p = 0.0$
- Activation: GELU, Layer norm: Pre-norm

Training Considerations

- Random initialization of memory $\mathbf{L}_0 \sim \mathcal{N}(0, 0.02^2)$
- Gradient clipping (max-norm = 1.0)
- Memory disabled ($P = 0$) for ablation

B.3.4 Dataset-wise In-Context Learning (TF_{icl})

Hyperparameters

- Transformer blocks: $N_{\text{icl}} = 12$
- Embedding dimension: $d_R = 512$
- Feedforward dimension: $d_{\text{ff}} = 1024$
- Max classes: $C_{\text{max}} = 10$
- Temperature: $\tau = 0.9$
- Dropout: $p = 0.0$
- Activation: GELU
- Layer norm: Pre-norm

Initialization

- Label encoder, Transformer, and decoder weights: Xavier/Glorot uniform
- Layer norm: $\gamma = 1$, $\beta = 0$

C Further Experiments

C.1 Extended Results

To further investigate the sources of Orion-MSP's performance gains, we analyze results across key dataset characteristics. All analyses partition datasets based on inherent properties rather than performance outcomes.

Dataset Size. Table 2 reports model performance aggregated by dataset size: Small ($< 1K$ samples), Medium (1K-10K), and Large (> 10). Performance trends reveal that Orion-MSP consistently performs well across small, medium, and large datasets. Classical ML models such as XGBoost excel on large datasets due to abundant training examples, achieving the highest ACC/F1 in the $> 10K$ sample category. Orion-MSP, however, maintains competitive performance across all size categories, outperforming most baselines on small and medium datasets. This demonstrates the ability of multi-scale sparse attention to generalize effectively in low-data regimes while scaling gracefully to larger datasets. TabPFN also performs strongly, particularly on medium-sized datasets, but Orion-MSP's consistent performance across size scales highlights the robustness of its hierarchical and sparse design.

Table 2: Performance variation by dataset size across all benchmark suites. Rank denotes the accuracy-based ranking per size category (lower is better). ACC = Accuracy; F1 = Weighted F1-score, averaged across datasets within each size category: Small (<1K samples), Medium (1K–10K), and Large (>10K).

Models	Small (<1K)			Medium (1K-10K)			Large (>10K)		
	Rank	ACC	F1	Rank	ACC	F1	Rank	ACC	F1
XGBoost	7.70	0.8168	0.7964	6.88	0.8363	0.8314	5.41	**0.8969**	**0.8920**
CatBoost	7.88	0.8124	0.7935	6.47	0.8340	0.8264	5.48	0.8797	0.8733
Random Forest	8.55	0.7988	0.8187	7.16	0.8285	0.8221	7.30	0.8694	0.8628
LightGBM	7.80	0.8143	0.7789	6.94	0.8314	0.8226	5.63	0.8827	0.8764
TabICL	6.04	0.8301	**0.8338**	4.77	0.8486	0.8398	4.61	0.8802	0.8743
OrionBiX	6.32	0.8330	0.8150	5.48	0.8348	0.8260	4.42	0.8729	0.8670
OrionMSP	5.93	0.8232	0.8194	**3.70**	0.8494	0.8402	**3.04**	0.8843	0.8768
TabPFN	6.50	0.8325	0.8131	3.81	**0.8557**	**0.8462**	5.73	0.8783	0.8713
Mitra	13.88	0.4334	0.3236	11.59	0.3600	0.2553	11.11	0.3837	0.2754
ContextTab	9.60	0.7578	0.7363	9.52	0.6210	0.5566	10.22	0.6388	0.5638
TabDPT	**5.48**	**0.8333**	0.8271	5.40	0.8424	0.8339	5.26	0.8831	0.8765

Feature Dimensionality. Table 3 presents performance trends across narrow (< 10 features), medium (10 - 100) and wide (> 100) datasets. When evaluating dataset width, Orion-MSP shows the highest accuracy on narrow datasets (<10 features) and strong performance on medium and wide datasets (10–100 and >100 features). This suggests that sparse multi-scale attention enables effective learning even in high-dimensional feature spaces, where dense models such as TabICL exhibit diminished scalability to high-dimensional feature spaces.

Table 3: Performance variation by feature dimensionality (dataset width) across all benchmark suites. Rank denotes the accuracy-based ranking averaged within each width category (lower is better). ACC = Accuracy; F1 = Weighted F1-score. Values are on a 0–1 scale (higher is better). Formatting: **1st place** ; 2nd place within each group.

Models	Narrow (<10)			Medium (10-100)			Wide (>100)		
	Rank	ACC	F1	Rank	ACC	F1	Rank	ACC	F1
XGBoost	6.77	0.8222	0.8159	6.90	0.8482	0.8410	**4.79**	0.9140	0.9039
CatBoost	5.63	0.8145	0.8067	6.88	0.8441	0.8344	5.50	**0.9157**	0.9084
Random Forest	7.15	0.8005	0.7044	7.44	0.8410	0.8235	7.52	0.9034	0.8936
LightGBM	6.15	0.8128	0.7907	6.92	0.8458	0.8326	7.47	0.8999	0.8908
TabICL	5.14	0.8208	0.8119	4.61	0.8627	0.8549	6.46	0.9101	0.8936
OrionBiX	4.64	0.8112	0.8043	5.46	0.8510	0.8417	6.73	0.8859	0.8849
OrionMSP	**3.76**	**0.8394**	**0.8314**	4.09	0.8572	0.8478	5.69	0.8860	0.8837
TabPFN	5.30	0.8187	0.8092	**4.07**	**0.8676**	**0.8589**	6.141	0.9129	**0.9111**
Mitra	11.25	0.3737	0.2683	11.84	0.3886	0.2781	13.03	0.2521	0.1497
ContextTab	9.52	0.6391	0.5719	9.59	0.6480	0.5843	10.97	0.6017	0.5651
TabDPT	4.66	0.8262	0.8189	5.45	0.8566	0.8483	7.23	0.8845	0.8820

Domain-specific Analysis. Domain-wise evaluation provides deeper insight into Orion-MSP's strengths (Table 4):

- Medical datasets: Orion-MSP achieves the highest ACC = 0.8045 and F1 = 0.7916, ranking second overall behind Orion-BiX. These datasets often involve hierarchical biological structures and complex interdependencies among features, which align naturally with Orion-

MSP's multi-scale representation. Fine-grained scales capture local dependencies, while coarser scales aggregate contextual information, leading to improved predictive accuracy.

- Finance datasets: Orion-MSP ranks first in mean rank (4.60), achieving ACC = 0.8158 and F1 = 0.8047. Financial datasets frequently involve layered dependencies between assets, instruments, and market indicators. Orion-MSP's cross-component memory allows information to propagate across scales, capturing global dependencies that standard dense transformers or classical ML models fail to exploit.

Table 4: Domain-specific performance for Medical and Finance datasets from the benchmark suites. Rank denotes the mean rank within each domain (lower is better). ACC = Accuracy; F1 = Weighted F1-score (0–1 scale, higher is better). Formatting: **1st place**; 2nd place within each group.

Models	Medical			Finance		
	Rank	ACC	F1	Rank	ACC	F1
XGBoost	6.32	0.7834	0.7669	6.62	0.7958	0.7885
RandomForest	6.38	0.7779	0.7752	7.32	0.8052	0.8001
CatBoost	6.36	0.7784	0.7594	5.82	0.8117	0.8015
LightGBM	5.32	0.7949	0.7614	6.17	0.8095	0.7974
TabICL	5.54	0.7819	0.7696	6.60	0.8125	0.7942
OrionBiX	**4.10**	0.7893	0.7759	5.39	**0.8206**	**0.8125**
OrionMSP	4.50	**0.8045**	**0.7916**	**4.60**	0.8158	0.8047
TabPFN	5.04	0.7984	0.7857	7.17	0.8094	0.7919
Mitra	10.77	0.3935	0.2863	13.67	0.5340	0.4250
ContextTab	8.66	0.6681	0.6129	11.25	0.7430	0.6834
TabDPT	6.86	0.7764	0.7641	8.00	0.8080	0.7960

Overall, domain-specific results highlight that Orion-MSP excels in high-dimensional, context-rich datasets, where hierarchical patterns and feature correlations are prevalent.

Deep Analysis and Interpretation

A detailed examination by dataset characteristics demonstrates why Orion-MSP's design is most effective under certain conditions:

- Class imbalance: Multi-scale sparse attention amplifies underrepresented patterns without overfitting to majority classes. Minority-class recognition improves substantially on datasets where the minority class constitutes less than 30% of the data. Balanced datasets show smaller gains, indicating that the hierarchical complexity is most beneficial in skewed settings.
- Hierarchical structure and cross-component memory: In domains such as healthcare and finance, datasets involve natural hierarchies and complex inter-feature relationships. Orion-MSP's multi-scale design allows it to reason at both fine-grained and coarse-grained levels. Sparse attention reduces computational cost and provides implicit regularization, mitigating overfitting in high-dimensional or correlated-feature settings. Cross-component memory further enables information exchange across scales without violating ICL safety, enhancing performance on context-dependent tasks.
- Computational efficiency: Linear attention complexity with respect to feature number and attention window size allows Orion-MSP to scale to high-dimensional tables. Memory usage grows proportionally with input dimensions, making the model practical for large real-world datasets, unlike dense attention alternatives with quadratic scaling.

C.2 Average Performance and Rankings

This section reports the average rank of all methods across dataset categories. Figure 4 shows the relative accuracy improvement (%) of each method over XGBoost for individual datasets across the three benchmarks: TALENT (Figure 4a), TabZilla (Figure 4b), and OpenML-CC18 (Figure 4c). Each point represents the per-dataset improvement, while the boxplots summarize the distribution, including the median, interquartile range, and whiskers. The dashed vertical line indicates parity with XGBoost (0%).

Across all three benchmarks, Orion-MSP consistently improves upon the strong XGBoost baseline. On TabZilla and OpenML-cc18, Orion-MSP achieves positive median relative accuracy with a compact interquartile range and few negative outliers, indicating both higher accuracy and greater reliability. On TALENT, Orion-MSP reaches parity in median performance but exhibits lower variance than most neural baselines.

Among classical boosted trees, LightGBM, CatBoost, and Random Forest cluster tightly around parity, showing comparable behavior. In contrast, tabular foundation models (TFMs), notably Mitra and ContextTab, exhibit pronounced negative shifts and high variance. TabPFN and TabICL perform competitively and occasionally outperform Orion-MSP on specific datasets, yet their broader variance and heavier left tails reveal less consistent behavior. Overall, Orion-MSP matches or surpasses their central performance and achieves the best mean–variance trade-off, confirming the benefits of our model design for tabular generalization.

To further quantify cross-dataset performance, we computed per-dataset ranks across all 11 methods (lower is better) for each benchmark, averaged them over datasets, and conducted a one-way Friedman test to assess overall differences. When significant, Nemenyi post-hoc tests were applied, and the resulting critical difference (CD) diagrams were plotted at $\alpha = 0.05$; methods connected by the CD bar are not significantly different.

As shown in Figure 5, Orion-MSP attains the best average rank on all three benchmarks—3.09 on TALENT, 3.32 on TabZilla, and 3.35 on OpenML-cc18—appearing as the leftmost method in each CD diagram. TabICL, TabPFN, and TabDPT follow closely, typically lagging by ≈ 0.5–1.5 rank points. Tree ensembles (XGBoost, LightGBM, CatBoost, Random Forest) occupy a middle tier with average ranks of 4.6–6.2, showing parity among themselves but a consistent gap to Orion-MSP and the stronger TFMs. Finally, ContextTab and Mitra form the rightmost group (ranks $\approx$ 9–10), confirming the underperformance seen in the improvement plots.

In summary, the CD diagrams corroborate our main finding: Orion-MSP is the top performer across diverse tabular benchmarks, outperforming tree ensembles on average and matching or exceeding pretrained tabular foundation models while maintaining a favorable significance profile.

C.3 Experimental Setting

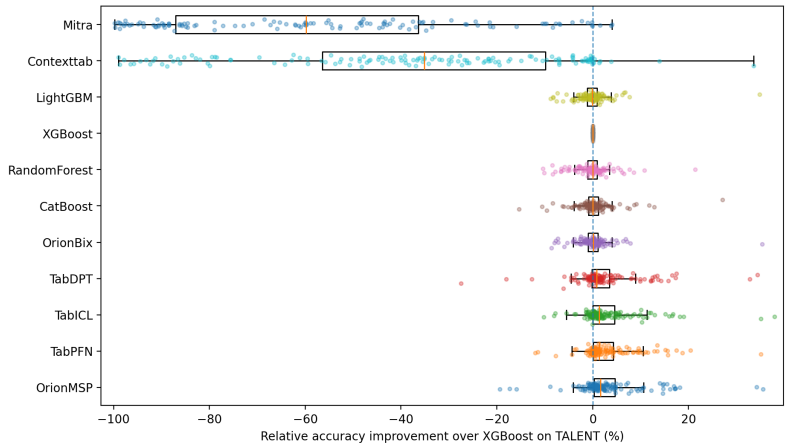
C.3.1 Benchmark Suites and Datasets.

Our experimental evaluation spans three widely recognized benchmark suites: TALENT [26] (181 automatically discovered classification datasets), OPENML-CC18 [2] (72 curated datasets), and TABZILLA [19] (36 heterogeneous tasks). Together, these benchmarks enable a comprehensive assessment across diverse tabular learning scenarios. In addition, we perform domain-specific evaluations in high-impact application areas such as healthcare and finance to examine the real-world relevance of our method. All experiments strictly follow the official dataset splits provided by each benchmark to ensure reproducibility and fairness.

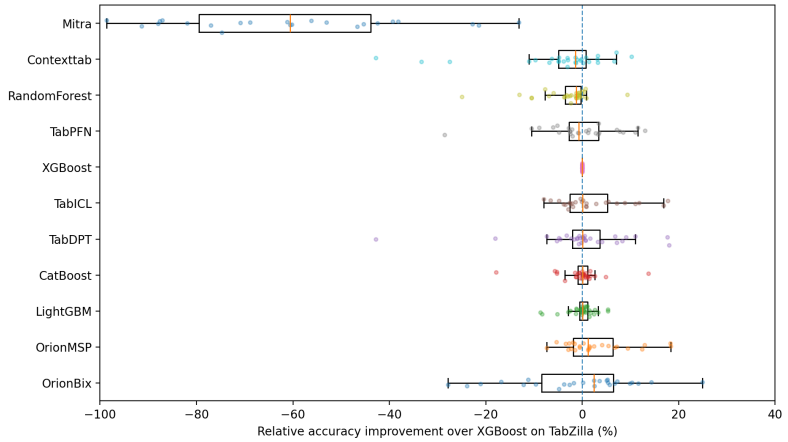
For consistency across model families, results are reported only on the intersection of datasets available to all evaluated models within each benchmark suite. This unified evaluation protocol ensures that observed performance differences arise from methodological advances rather than variations in dataset coverage. After filtering, our evaluation encompasses 154 of 181 datasets from TALENT, 63 of 72 from OpenML-CC18, and 27 of 36 from TabZilla. A small number of datasets were excluded due to out-of-memory (OOM) errors or CUDA-related issues, primarily affecting TabPFN and ContextTab-based architectures even on H200 GPUs.

C.3.2 Models and Baselines.

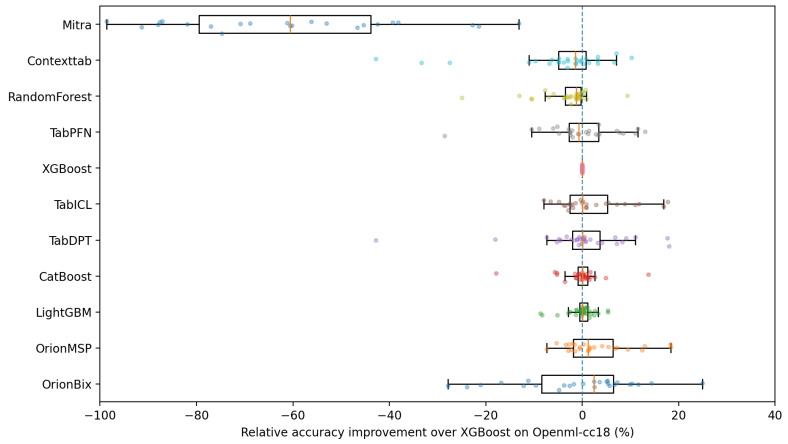
We compare our model with six state-of-the-art tabular foundation models: TABPFN [9], TABICL [21], ORIONBIX, MITRA, CONTEXTTAB [23], and TABDPT [18]. In addition, we include established `traditional baselines` using autogloun [4] such as XGBOOST, LIGHTGBM, CATBOOST, and RANDOM FOREST as strong reference models for comparison.



(a) Relative accuracy improvement over XGBoost on TALENT Benchmark

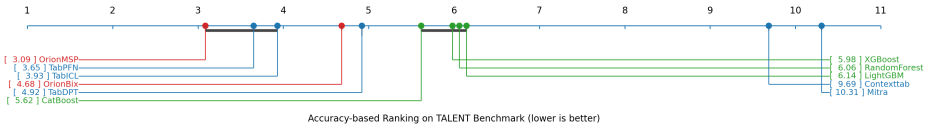


(b) Relative accuracy improvement over XGBoost on TabZilla Benchmark

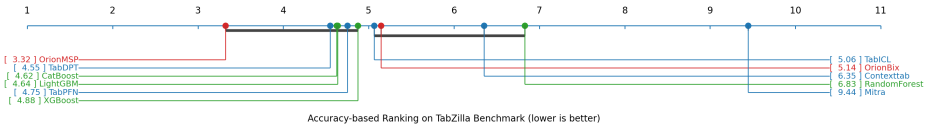


(c) Relative accuracy improvement over XGBoost on OPENML-CC18 Benchmark

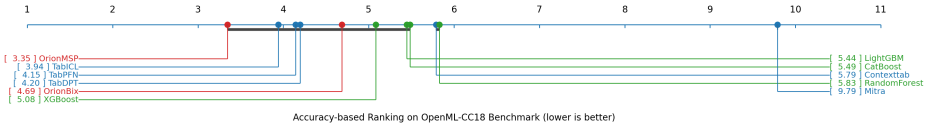
Figure 4: Relative accuracy improvement over XGBoost on three benchmarks.



(a) Relative accuracy improvement over XGBoost on TALENT Benchmark



(b) Relative accuracy improvement over XGBoost on TabZilla Benchmark



(c) Relative accuracy improvement over XGBoost on OPENML-CC18 Benchmark

Figure 5

C.3.3 Hardware Configuration.

Experiments are executed on NVIDIA L40S GPUs, with H200 GPUs used for memory-intensive cases. This infrastructure ensures consistent execution across all experiments while handling the computational demands of large transformer-based models.

C.3.4 Evaluation Metrics.

Our evaluation considers two complementary aspects:

Performance. We measure predictive capability using standard classification metrics—Accuracy (ACC), AUC-ROC, and weighted F1-score (F1)—computed across the benchmark suites TALENT, OpenML-CC18, and TabZilla. These benchmarks encompass datasets with diverse characteristics, including varying sample sizes, feature dimensionalities, and class balance, allowing a comprehensive assessment of model generalization. It is important to clarify how MEAN RANK values are derived. Within each benchmark suite, models are ranked by accuracy on every dataset (lower rank = better performance), and these per-dataset ranks are averaged to obtain the overall mean rank. Thus, a lower mean rank indicates stronger and more consistent performance across datasets, rather than the highest score on any single task. While absolute metrics (ACCURACY, F1) reflect peak task-level performance, mean rank provides a normalized measure of cross-dataset generalization consistency.

Scalability. We further analyze model robustness as dataset complexity increases by examining performance trends with respect to sample size, feature dimensionality, and class imbalance. This analysis uses the same benchmark datasets, aggregated along these axes to reveal systematic scalability behaviors and guide practical model selection.

C.4 Datasets

Full details of all datasets and benchmarks are summarized in below, with their row and column distributions visualized in Figure 6.

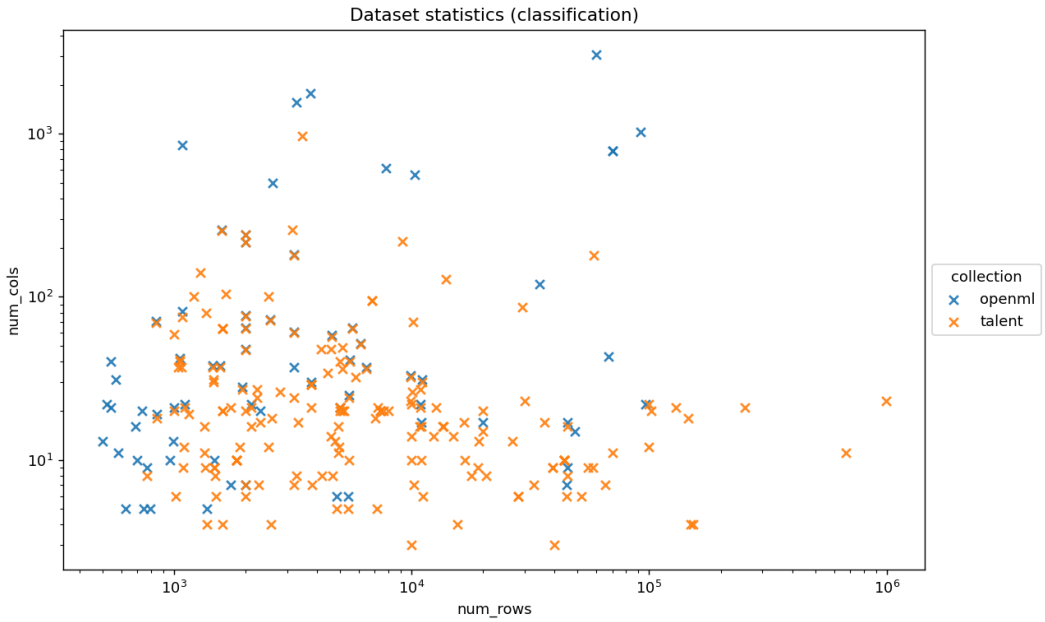


Figure 6: Column and row distribution of the evaluated datasets.

OpenML-CC18 Benchmark Datasets

Table 5 lists all datasets from the OpenML-CC18 benchmark suite used in our evaluation.

TALENT Benchmark Datasets

Table 6 lists all datasets from the TALENT benchmark suite used in our evaluation.

TabZilla Benchmark Datasets

Table 7 lists all datasets from the TabZilla benchmark suite. TabZilla uses OpenML dataset IDs, and these datasets are specifically selected for evaluating neural network performance on tabular data.

Table 5: OpenML-CC18 benchmark datasets (72 datasets).

Benchmark	Dataset Name	Domain	Samples	Features	Classes	Task Type	Used
OpenML	OpenML-ID-3	Other	3196	37	2	binclass	Yes
OpenML	OpenML-ID-6	Handwriting	20000	17	26	multiclass	No
OpenML	OpenML-ID-11	Other	625	5	3	multiclass	Yes
OpenML	OpenML-ID-12	Other	2000	217	10	multiclass	Yes
OpenML	OpenML-ID-14	Other	2000	77	10	multiclass	Yes
OpenML	OpenML-ID-15	Healthcare	699	10	2	binclass	Yes
OpenML	OpenML-ID-16	Other	2000	65	10	multiclass	Yes
OpenML	OpenML-ID-18	Other	2000	7	10	multiclass	Yes
OpenML	OpenML-ID-22	Other	2000	48	10	multiclass	Yes
OpenML	OpenML-ID-23	Healthcare	1473	10	3	multiclass	Yes
OpenML	OpenML-ID-28	Handwriting	5620	65	10	multiclass	Yes
OpenML	OpenML-ID-29	Finance	690	16	2	binclass	Yes
OpenML	OpenML-ID-31	Finance	1000	21	2	binclass	Yes
OpenML	OpenML-ID-32	Handwriting	10992	17	10	multiclass	Yes
OpenML	OpenML-ID-37	Healthcare	768	9	2	binclass	Yes
OpenML	OpenML-ID-38	Healthcare	3772	30	2	binclass	Yes
OpenML	OpenML-ID-44	Other	4601	58	2	binclass	Yes
OpenML	OpenML-ID-46	Other	3190	61	3	multiclass	Yes

Continued on next page

Table 5: Details of OpenML-CC18 benchmark datasets.

Benchmark	Dataset Name	Domain	Samples	Features	Classes	Task Type	Used
OpenML	OpenML-ID-50	Other	958	10	2	binclass	Yes
OpenML	OpenML-ID-54	Other	846	19	4	multiclass	Yes
OpenML	OpenML-ID-151	Other	45312	9	2	binclass	Yes
OpenML	OpenML-ID-182	Other	6430	37	6	multiclass	Yes
OpenML	OpenML-ID-188	Other	736	20	5	multiclass	Yes
OpenML	OpenML-ID-300	Other	7797	618	26	multiclass	No
OpenML	OpenML-ID-307	Other	990	13	11	multiclass	No
OpenML	OpenML-ID-458	Other	841	71	4	multiclass	Yes
OpenML	OpenML-ID-469	Healthcare	797	5	6	multiclass	Yes
OpenML	OpenML-ID-554	Other	70000	785	10	multiclass	Yes
OpenML	OpenML-ID-1049	Other	1458	38	2	binclass	Yes
OpenML	OpenML-ID-1050	Other	1563	38	2	binclass	Yes
OpenML	OpenML-ID-1053	Other	10885	22	2	binclass	Yes
OpenML	OpenML-ID-1063	Other	522	22	2	binclass	Yes
OpenML	OpenML-ID-1067	Other	2109	22	2	binclass	Yes
OpenML	OpenML-ID-1068	Other	1109	22	2	binclass	Yes
OpenML	OpenML-ID-1461	Finance	45211	17	2	binclass	Yes
OpenML	OpenML-ID-1462	Finance	1372	5	2	binclass	Yes
OpenML	OpenML-ID-1464	Healthcare	748	5	2	binclass	Yes
OpenML	OpenML-ID-1468	Other	1080	857	9	multiclass	Yes
OpenML	OpenML-ID-1475	Other	6118	52	6	multiclass	Yes
OpenML	OpenML-ID-1478	Other	10299	562	6	multiclass	Yes
OpenML	OpenML-ID-1480	Healthcare	583	11	2	binclass	Yes
OpenML	OpenML-ID-1485	Other	2600	501	2	binclass	Yes
OpenML	OpenML-ID-1486	Other	34465	119	2	binclass	No
OpenML	OpenML-ID-1487	Other	2534	73	2	binclass	Yes
OpenML	OpenML-ID-1489	Other	5404	6	2	binclass	Yes
OpenML	OpenML-ID-1494	Other	1055	42	2	binclass	Yes
OpenML	OpenML-ID-1497	Other	5456	25	4	multiclass	Yes
OpenML	OpenML-ID-1501	Other	1593	257	10	multiclass	Yes
OpenML	OpenML-ID-1510	Other	569	31	2	binclass	Yes
OpenML	OpenML-ID-1590	Other	48842	15	2	binclass	Yes
OpenML	OpenML-ID-4134	Other	3751	1777	2	binclass	No
OpenML	OpenML-ID-4534	Other	11055	31	2	binclass	Yes
OpenML	OpenML-ID-4538	Other	9873	33	5	multiclass	Yes
OpenML	OpenML-ID-6332	Other	540	40	2	binclass	Yes
OpenML	OpenML-ID-23381	Retail	500	13	2	binclass	Yes
OpenML	OpenML-ID-23517	Other	96320	22	2	binclass	No
OpenML	OpenML-ID-40499	Other	5500	41	11	multiclass	No
OpenML	OpenML-ID-40668	Games	67557	43	3	multiclass	No
OpenML	OpenML-ID-40670	Other	3186	181	3	multiclass	Yes
OpenML	OpenML-ID-40701	Other	5000	21	2	binclass	Yes
OpenML	OpenML-ID-40923	Other	92000	1025	46	multiclass	No
OpenML	OpenML-ID-40927	Handwriting	60000	3073	10	multiclass	No
OpenML	OpenML-ID-40966	Other	1080	82	8	multiclass	No
OpenML	OpenML-ID-40975	Other	1728	7	4	multiclass	Yes
OpenML	OpenML-ID-40978	Other	3279	1559	2	binclass	Yes
OpenML	OpenML-ID-40979	Other	2000	241	10	multiclass	Yes
OpenML	OpenML-ID-40982	Other	1941	28	7	multiclass	Yes
OpenML	OpenML-ID-40983	Other	4839	6	2	binclass	Yes
OpenML	OpenML-ID-40984	Other	2310	20	7	multiclass	Yes
OpenML	OpenML-ID-40994	Other	540	21	2	binclass	Yes
OpenML	OpenML-ID-40996	Other	70000	785	10	multiclass	No
OpenML	OpenML-ID-41027	Games	44819	7	3	multiclass	Yes

Table 6: TALENT benchmark datasets (auto-discovered, multiple domains).

Benchmark	Dataset Name	Domain	Samples	Features	Classes	Task Type	Used
TALENT	ASP-POTASSCO-class	Other	1294	141	11	multiclass	Yes
TALENT	Amazon_employee_access	Other	32769	7	2	binclass	Yes
TALENT	BLE_RSSI__Indoor_localization	Other	9984	3	3	multiclass	Yes
TALENT	BNG(breast-w)	Healthcare	39366	9	2	binclass	Yes
TALENT	BNG(cmc)	Other	55296	9	3	multiclass	Yes
TALENT	BNG(tic-tac-toe)	Other	39366	9	2	binclass	Yes
TALENT	Bank_Customer_Churn	Finance	10000	10	2	binclass	Yes
TALENT	Basketball_c	Retail	1340	11	2	binclass	Yes
TALENT	CDC_Diabetes_Health	Healthcare	253680	21	2	binclass	Yes
TALENT	California-Housing-Class	Other	20640	8	2	binclass	No
TALENT	Cardiovascular-Disease	Healthcare	70000	11	2	binclass	Yes
TALENT	Click_prediction_small	Other	39948	3	2	binclass	Yes
TALENT	Credit_c	Finance	100000	22	3	multiclass	Yes
TALENT	Customer_Personality_Analysis	Retail	2240	24	2	binclass	Yes
TALENT	DataScience_Kiva_Crowdfunding	Other	671205	11	4	multiclass	No
TALENT	Diabetic_Retinopathy_Debrecen	Healthcare	1151	19	2	binclass	Yes
TALENT	E-CommereShippingData	Other	10999	10	2	binclass	Yes
TALENT	Employee	Other	4653	8	2	binclass	Yes
TALENT	FICO-HELOC-cleaned	Other	9871	23	2	binclass	Yes
TALENT	FOREX_audcad-day-High	Finance	1834	10	2	binclass	No
TALENT	FOREX_audcad-hour-High	Finance	43825	10	2	binclass	No
TALENT	FOREX_audchf-day-High	Finance	1833	10	2	binclass	No
TALENT	FOREX_audjpy-day-High	Finance	1832	10	2	binclass	No
TALENT	FOREX_audjpy-hour-High	Finance	43825	10	2	binclass	No
TALENT	FOREX_audsgd-hour-High	Finance	43825	10	2	binclass	No
TALENT	FOREX_audusd-hour-High	Finance	43825	10	2	binclass	No
TALENT	FOREX_cadjpy-day-High	Finance	1834	10	2	binclass	No
TALENT	FOREX_cadjpy-hour-High	Finance	43825	10	2	binclass	No
TALENT	Firm-Teacher_Clave-Direction	Other	10800	16	4	multiclass	Yes
TALENT	Fitness_Club_c	Other	1500	6	2	binclass	Yes
TALENT	GAMETES_Epistasis_2-Way	Games	1600	20	2	binclass	Yes
TALENT	GAMETES_Heterogeneity	Games	1600	20	2	binclass	Yes
TALENT	Gender_Gap_in_Spanish	Other	4746	13	3	multiclass	Yes
TALENT	GesturePhaseSegmentation	Other	9873	32	5	multiclass	Yes
TALENT	HR_Analytics_Job_Change	Other	19158	13	2	binclass	Yes
TALENT	IBM_HR_Analytics	Other	1470	31	2	binclass	Yes
TALENT	INNHotelsGroup	Other	36275	17	2	binclass	Yes
TALENT	Indian_pines	Other	9144	220	8	multiclass	Yes
TALENT	JapaneseVowels	Other	9961	14	9	multiclass	Yes
TALENT	KDDCup09_upselling	Other	5128	49	2	binclass	Yes
TALENT	MIC	Other	1649	104	2	binclass	Yes
TALENT	MagicTelescope	Other	19020	9	2	binclass	Yes
TALENT	Marketing_Campaign	Finance	2240	27	2	binclass	Yes
TALENT	Mobile_Price_Classification	Telcom	2000	20	4	multiclass	Yes
TALENT	National_Health_and_Nutrition	Healthcare	2278	7	2	binclass	Yes
TALENT	PhishingWebsites	Other	11055	30	2	binclass	Yes
TALENT	PieChart3	Other	1077	37	2	binclass	Yes
TALENT	Pima_Indians_Diabetes	Healthcare	768	8	2	binclass	Yes
TALENT	PizzaCutter3	Other	1043	37	2	binclass	Yes
TALENT	Pumpkin_Seeds	Other	2500	12	2	binclass	Yes
TALENT	QSAR_biodegradation	Healthcare	1054	41	2	binclass	Yes
TALENT	Rain_in_Australia	Other	145460	18	3	multiclass	No
TALENT	SDSS17	Other	100000	12	3	multiclass	Yes
TALENT	Satellite	Other	5100	36	2	binclass	Yes
TALENT	Smoking_and_Drinking	Other	991346	23	2	binclass	No
TALENT	Telecom_Churn_Dataset	Telcom	3333	17	2	binclass	Yes
TALENT	UJI_Pen_Characters	Other	1364	80	35	multiclass	Yes
TALENT	Water_Quality_and_Potability	Manufacturing	3276	8	2	binclass	Yes
TALENT	Wilt	Other	4821	5	2	binclass	Yes

Continued on next page

Table 6: Details of TALENT benchmark datasets.

Benchmark	Dataset Name	Domain	Samples	Features	Classes	Task Type	Used
TALENT	abalone	Other	4177	8	3	multiclass	Yes
TALENT	accelerometer	Other	153004	4	4	multiclass	No
TALENT	ada	Other	4147	48	2	binclass	Yes
TALENT	ada_agnostic	Other	4562	48	2	binclass	Yes
TALENT	ada_prior	Other	4562	14	2	binclass	Yes
TALENT	airlines_seed_0_nrows	Telcom	2000	7	2	binclass	Yes
TALENT	allbp	Other	3772	29	3	multiclass	Yes
TALENT	allrep	Other	3772	29	4	multiclass	Yes
TALENT	analcatdata_authorship	Other	841	69	4	multiclass	Yes
TALENT	artificial-characters	Other	10218	7	10	multiclass	Yes
TALENT	autoUniv-au4-2500	Other	2500	100	3	multiclass	Yes
TALENT	autoUniv-au7-1100	Other	1100	12	5	multiclass	Yes
TALENT	bank	Finance	45211	16	2	binclass	Yes
TALENT	banknote_authentication	Finance	1372	4	2	binclass	Yes
TALENT	baseball	Other	1340	16	3	multiclass	Yes
TALENT	car-evaluation	Other	1728	21	4	multiclass	Yes
TALENT	churn	Finance	5000	20	2	binclass	Yes
TALENT	cmc	Other	1473	9	3	multiclass	Yes
TALENT	company_bankruptcy_prediction	Finance	6819	95	2	binclass	Yes
TALENT	compass	Other	16644	17	2	binclass	Yes
TALENT	contraceptive_method_choice	Other	1473	9	3	multiclass	Yes
TALENT	credit	Finance	16714	10	2	binclass	Yes
TALENT	customer_satisfaction_in_airline	Retail	129880	21	2	binclass	No
TALENT	dabetes_130-us_hospitals	Healthcare	101766	20	2	binclass	Yes
TALENT	default_of_credit_card_clients	Finance	30000	23	2	binclass	Yes
TALENT	delta_ailerons	Other	7129	5	2	binclass	Yes
TALENT	dis	Other	3772	29	2	binclass	Yes
TALENT	dna	Healthcare	3186	180	3	multiclass	Yes
TALENT	drug_consumption	Healthcare	1884	12	7	multiclass	Yes
TALENT	dry_bean_dataset	Other	13611	16	7	multiclass	Yes
TALENT	eeg-eye-state	Other	14980	14	2	binclass	Yes
TALENT	electricity	Manufacturing	45312	8	2	binclass	Yes
TALENT	estimation_of_obesity_levels	Other	2111	16	7	multiclass	Yes
TALENT	eye_movements	Healthcare	10936	27	3	multiclass	Yes
TALENT	eye_movements_bin	Healthcare	7608	20	2	binclass	Yes
TALENT	first-order-theorem-proving	Other	6118	51	6	multiclass	Yes
TALENT	gas-drift	Other	13910	128	6	multiclass	Yes
TALENT	gina_agnostic	Other	3468	970	2	binclass	Yes
TALENT	golf_play_dataset_extended	Other	1095	9	2	binclass	Yes
TALENT	heloc	Other	10000	22	2	binclass	Yes
TALENT	hill-valley	Other	1212	100	2	binclass	No
TALENT	house_16H	Other	13488	16	2	binclass	Yes
TALENT	htru	Other	17898	8	2	binclass	Yes
TALENT	ibm-employee-performance	Other	1470	30	2	binclass	Yes
TALENT	in_vehicle_coupon_recos	Retail	12684	21	2	binclass	Yes
TALENT	internet_firewall	Other	65532	7	4	multiclass	Yes
TALENT	internet_usage	Other	10108	70	46	multiclass	Yes
TALENT	jm1	Other	10885	21	2	binclass	No
TALENT	jungle_chess_2pcs_raw	Other	44819	6	3	multiclass	Yes
TALENT	kc1	Other	2109	21	2	binclass	No
TALENT	kdd_ipums_la_97-small	Other	5188	20	2	binclass	Yes
TALENT	kr-vs-k	Other	28056	6	18	multiclass	Yes
TALENT	kropt	Other	28056	6	18	multiclass	Yes
TALENT	led24	Other	3200	24	10	multiclass	Yes
TALENT	led7	Other	3200	7	10	multiclass	Yes
TALENT	letter	Other	20000	15	26	multiclass	Yes
TALENT	madeline	Other	3140	259	2	binclass	Yes
TALENT	mammography	Other	11183	6	2	binclass	Yes
TALENT	maternal_health_risk	Healthcare	1014	6	3	multiclass	Yes

Continued on next page

Table 6: Details of TALENT benchmark datasets.

Benchmark	Dataset Name	Domain	Samples	Features	Classes	Task Type	Used
TALENT	mfeat-factors	Other	2000	216	10	multiclass	Yes
TALENT	mfeat-fourier	Other	2000	76	10	multiclass	Yes
TALENT	mfeat-karhunen	Other	2000	64	10	multiclass	Yes
TALENT	mfeat-morphological	Other	2000	6	10	multiclass	Yes
TALENT	mfeat-pixel	Other	2000	240	10	multiclass	Yes
TALENT	mfeat-zernike	Other	2000	47	10	multiclass	Yes
TALENT	mice_protein_expression	Other	1080	75	8	multiclass	Yes
TALENT	microaggregation2	Other	20000	20	5	multiclass	Yes
TALENT	mobile_c36_oversampling	Telcom	51760	6	2	binclass	Yes
TALENT	mozilla4	Other	15545	4	2	binclass	Yes
TALENT	naticusdroid+android	Healthcare	29332	86	2	binclass	Yes
TALENT	national-longitudinal-survey-binary	Other	4908	16	2	binclass	Yes
TALENT	okcupid_stem	Other	26677	13	3	multiclass	Yes
TALENT	one-hundred-plants-margin	Other	1600	64	100	multiclass	Yes
TALENT	one-hundred-plants-shape	Other	1600	64	100	multiclass	Yes
TALENT	one-hundred-plants-texture	Other	1599	64	100	multiclass	Yes
TALENT	online_shoppers	Retail	12330	14	2	binclass	No
TALENT	optdigits	Other	5620	64	10	multiclass	Yes
TALENT	ozone-level-8hr	Other	2534	72	2	binclass	Yes
TALENT	page-blocks	Other	5473	10	5	multiclass	Yes
TALENT	pc1	Other	1109	21	2	binclass	No
TALENT	pc3	Other	1563	37	2	binclass	No
TALENT	pc4	Other	1458	37	2	binclass	No
TALENT	pendigits	Other	10992	16	10	multiclass	Yes
TALENT	phoneme	Other	5404	5	2	binclass	Yes
TALENT	pol	Other	10082	26	2	binclass	Yes
TALENT	predict_students_dropout	Other	4424	34	3	multiclass	Yes
TALENT	qsar	Other	1055	40	2	binclass	Yes
TALENT	rice_cammeo_and_osmancik	Other	3810	7	2	binclass	Yes
TALENT	ringnorm	Other	7400	20	2	binclass	Yes
TALENT	rl	Other	4970	12	2	binclass	No
TALENT	satimage	Other	6430	36	6	multiclass	Yes
TALENT	segment	Other	2310	17	7	multiclass	Yes
TALENT	seismic+bumps	Other	2584	18	2	binclass	Yes
TALENT	semeion	Other	1593	256	10	multiclass	No
TALENT	shuttle	Other	58000	9	7	multiclass	Yes
TALENT	spambase	Other	4601	57	2	binclass	Yes
TALENT	splice	Other	3190	60	3	multiclass	Yes
TALENT	sports_articles_for_objectivity	Other	1000	59	2	binclass	Yes
TALENT	statlog	Other	1000	20	2	binclass	Yes
TALENT	steel_plates_faults	Other	1941	27	7	multiclass	Yes
TALENT	sylvine	Other	5124	20	2	binclass	Yes
TALENT	taiwanese_bankruptcy	Finance	6819	95	2	binclass	Yes
TALENT	telco-customer-churn	Telcom	7043	18	2	binclass	Yes
TALENT	texture	Other	5500	40	11	multiclass	Yes
TALENT	thyroid	Healthcare	7200	21	3	multiclass	Yes
TALENT	thyroid-ann	Healthcare	3772	21	3	multiclass	Yes
TALENT	thyroid-dis	Healthcare	2800	26	5	multiclass	Yes
TALENT	turiye_student_evaluation	Other	5820	32	5	multiclass	Yes
TALENT	twonorm	Other	7400	20	2	binclass	Yes
TALENT	vehicle	Other	846	18	4	multiclass	Yes
TALENT	volkert	Other	58310	180	10	multiclass	Yes
TALENT	walking-activity	Other	149332	4	22	multiclass	No
TALENT	wall-robot-navigation	Other	5456	24	4	multiclass	Yes
TALENT	water_quality	Manufacturing	7996	20	2	binclass	Yes
TALENT	waveform-5000	Other	5000	40	3	multiclass	Yes
TALENT	waveform_database_generator	Other	4999	21	3	multiclass	Yes
TALENT	waveform_database_generator-v2	Other	5000	21	3	multiclass	Yes

Continued on next page

Table 6: Details of TALENT benchmark datasets.

Benchmark	Dataset Name	Domain	Samples	Features	Classes	Task Type	Used
TALENT	website_phishing	Other	1353	9	3	multiclass	Yes
TALENT	wine	Manufacturing	2554	4	2	binclass	No
TALENT	wine-quality-red	Manufacturing	1599	4	6	multiclass	Yes
TALENT	wine-quality-white	Manufacturing	4898	11	7	multiclass	Yes
TALENT	yeast	Other	1484	8	10	multiclass	Yes

Table 7: TabZilla benchmark datasets (36 datasets via OpenML).

Benchmark	Dataset Name	Domain	Samples	Features	Classes	Task Type	Used
TabZilla	OpenML-ID-999	Health	226	70	2	binclass	Yes
TabZilla	OpenML-ID-10	Health	148	19	4	multiclass	Yes
TabZilla	OpenML-ID-11	Other	625	5	3	multiclass	Yes
TabZilla	OpenML-ID-14	Other	2000	77	10	multiclass	Yes
TabZilla	OpenML-ID-22	Other	2000	48	10	multiclass	Yes
TabZilla	OpenML-ID-29	Finance	690	16	2	binclass	Yes
TabZilla	OpenML-ID-27	Health	368	23	2	binclass	Yes
TabZilla	OpenML-ID-31	Finance	1000	21	2	binclass	Yes
TabZilla	OpenML-ID-46	Other	3190	61	3	multiclass	Yes
TabZilla	OpenML-ID-54	Other	846	19	4	multiclass	Yes
TabZilla	OpenML-ID-333	Other	556	7	2	binclass	Yes
TabZilla	OpenML-ID-1067	Other	2109	22	2	binclass	Yes
TabZilla	OpenML-ID-1468	Other	1080	857	9	multiclass	Yes
TabZilla	OpenML-ID-1494	Other	1055	42	2	binclass	Yes
TabZilla	OpenML-ID-43973	Other	3172	6	2	binclass	Yes
TabZilla	OpenML-ID-1043	Other	4562	49	2	binclass	Yes
TabZilla	OpenML-ID-43945	Other	38474	9	2	binclass	Yes
TabZilla	OpenML-ID-1486	Other	34465	119	2	binclass	No
TabZilla	OpenML-ID-42825	Other	8378	123	–	–	No
TabZilla	OpenML-ID-4538	Other	9873	33	5	multiclass	Yes
TabZilla	OpenML-ID-23512	Other	98050	29	2	binclass	No
TabZilla	OpenML-ID-4134	Other	3751	1777	2	binclass	Yes
TabZilla	OpenML-ID-470	Other	672	10	2	binclass	No
TabZilla	OpenML-ID-1493	Other	1599	65	100	multiclass	Yes
TabZilla	OpenML-ID-1459	Other	10218	8	10	multiclass	Yes
TabZilla	OpenML-ID-41027	Games	44819	7	3	multiclass	Yes
TabZilla	OpenML-ID-40981	Other	690	15	2	binclass	Yes
TabZilla	OpenML-ID-934	Other	1156	6	2	binclass	Yes
TabZilla	OpenML-ID-1565	Health	294	14	5	multiclass	Yes
TabZilla	OpenML-ID-41150	Other	130064	51	2	binclass	No
TabZilla	OpenML-ID-41159	Other	20000	4297	2	binclass	No
TabZilla	OpenML-ID-846	Other	16599	19	2	binclass	Yes
TabZilla	OpenML-ID-1169	Other	539383	8	2	binclass	No
TabZilla	OpenML-ID-41147	Other	425240	79	2	binclass	Yes
TabZilla	OpenML-ID-41143	Other	2984	145	2	binclass	Yes
TabZilla	OpenML-ID-1567	Other	1025009	11	10	multiclass	No