NeurIDA: Dynamic Modeling for Effective In-Database Analytics

Lingze Zeng
lingze@comp.nus.edu.sg
National University of Singapore

Shaofeng Cai
shaofeng@comp.nus.edu.sg
National University of Singapore

Naili Xing*
xingnl@comp.nus.edu.sg
National University of Singapore

Jiaqi Zhu
jiaqi77@nus.edu.sg
National University of Singapore

Gang Chen
cg@zju.edu.cn
Zhejiang University

Peng Lu
peng.lu@zju.edu.cn
Zhejiang University

Jian Pei
j.pei@duke.edu
Duke University

Beng Chin Ooi
ooibc@zju.edu.cn
Zhejiang University

ABSTRACT

Relational Database Management Systems (RDBMS) manage complex, interrelated data and support a broad spectrum of analytical tasks. With the growing demand for predictive analytics, the deep integration of machine learning (ML) into RDBMS has become critical. However, a fundamental challenge hinders this evolution: conventional ML models are static and task-specific, whereas RDBMS environments are dynamic and must support diverse analytical queries. Each analytical task entails constructing a bespoke pipeline from scratch, which incurs significant development overhead and hence limits the wide adoption of ML in analytics.

We present NeurIDA, an autonomous end-to-end system for in-database analytics that dynamically "tweaks" the best available base model to better serve a given analytical task. In particular, we propose a novel paradigm of *dynamic in-database modeling* to pre-train a composable base model architecture over the relational data. Upon receiving a task, NeurIDA formulates the task and data profile to dynamically select and configure relevant components from the pool of base models and shared model components for prediction. For a friendly user experience, NeurIDA supports natural language queries; it interprets user intent to construct structured task profiles and generates analytical reports with dedicated LLM agents. By design, NeurIDA enables ease-of-use and yet effective and efficient in-database AI analytics. Extensive experimental studies show that NeurIDA consistently delivers up to 12% improvement in AUC-ROC and 25% relative reduction in MAE across ten tasks on five real-world datasets.

PVLDB Reference Format:
Lingze Zeng, Shaofeng Cai, Naili Xing, Jiaqi Zhu, Gang Chen, Peng Lu, Jian Pei, and Beng Chin Ooi. NeurIDA: Dynamic Modeling for Effective In-Database Analytics. PVLDB, 19(9): 2452 - 2465, 2026.
doi:10.14778/3819518.3819563

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/nusdbsystem/NeurIDA.

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 19, No. 9 ISSN 2150-8097.
doi:10.14778/3819518.3819563

1 INTRODUCTION

Relational Database Management Systems (RDBMS) form the cornerstone of modern data analytics, enabling complex analytical workflows via structured query language (SQL) [11, 32, 45, 49]. The deep integration of machine learning (ML) represents the next frontier that promises to transform these systems from passive data repositories into proactive analytical engines [9, 16, 24, 30]. However, while data-centric systems increasingly embed ML models within RDBMS, a fundamental paradigm gap between the static nature of ML and the dynamic environment of RDBMS has hindered this evolution, preventing the full realization of in-database AI-driven analytics [38, 60].

The root of this challenge lies in the inherent divide in their designs [61]. RDBMS are built for dynamism, designed to support a diverse and evolving range of analytical tasks on continuously updating data [26, 51]. In contrast, conventional ML models are inherently rigid, typically trained for a single, predefined prediction task on a static dataset snapshot and remain fixed after deployment [27, 54]. Consequently, when a new analytical task is issued, an existing model often cannot be readily adapted. For instance, in an e-commerce database, a model built to predict customer churn from user behavior is ill-suited to forecasting product returns based on item reviews [14]. This model-level inflexibility necessitates the costly and inefficient practice of building a bespoke data-to-model pipeline from scratch for each new analytical requirement.

Current approaches to bridge this gap typically fall into two main strategies. (1) Retraining a general-purpose model for each new task: This approach is resource-intensive and it undermines integrated analytics by requiring manual, external data processing that disrupts the in-database workflow [31, 41, 48]. (2) Maintaining a pool of pre-trained models and selecting the "best fit" at runtime: This approach is constrained by the inherent inflexibility of the candidate models and the non-trivial challenge of curating a relevant pool amid continuously evolving data [33, 36, 56, 63, 64]. Clearly, both strategies are fundamentally limited by their reliance on static models. They lack the capacity for fine-grained "dynamic" customization based on the specific semantics of an analytical query, such as its accessed tables, filter predicates, and therefore fail to consistently deliver optimal performance across diverse tasks [59].

Overcoming these limitations demands a new paradigm for ML-RDBMS integration that moves beyond static, embedded ML models towards a truly dynamic architecture. This requires a system capable of (1) dynamically constructing models tailored to individual analytical tasks, (2) operating directly over native relational data to

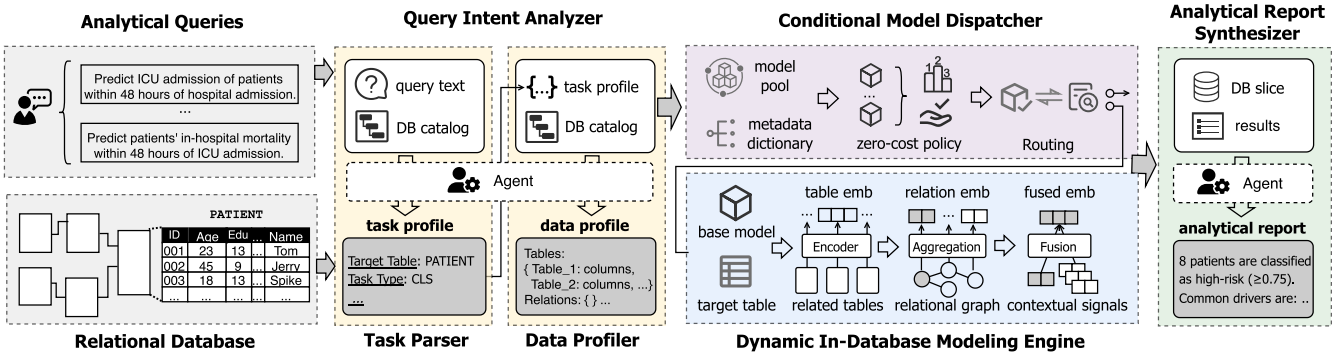


Figure 1: The workflow and key components of NeurIDA.

eliminate complex preprocessing, and (3) providing a unified interface for seamless task formulation and result interpretation. Such a system would ultimately render analytical workflows autonomous, efficient, and user-friendly.

In this paper, we present NeurIDA, a **Neur**al **I**n-**D**atabase **A**nalytics system that realizes these requirements through a novel paradigm of *dynamic in-database modeling*. To autonomously handle diverse analytical tasks end-to-end, as illustrated in Figure 1, the workflow of NeurIDA is orchestrated by four key components: First, given a natural language query (NLQ), the Query Intent Analyzer provides the unified interface that parses user intent to formulate the task, producing a structured task and data profile. The Conditional Model Dispatcher then selects the best available base model and conditionally invokes dynamic modeling for model augmentation. Next, the Dynamic In-database Modeling Engine (DIME) operates on the relational database to retrieve the required data and execute the model prediction. It either directly deploys the selected base model for efficiency or dynamically constructs a bespoke model to augment the base model for better serving the given task. Finally, the Analytical Report Synthesizer handles result interpretation, synthesizing the prediction results into a comprehensive technical report delivered to the user.

At the core of NeurIDA is DIME, which shifts modeling from a static design to a composable paradigm. When model augmentation is invoked, DIME dynamically constructs a bespoke model at query time using the selected base model and shared modules. This dynamic modeling is a multi-stage process explicitly guided by the task: First, DIME generates tuple embeddings via *base table embedding* to capture individual-table semantics for the retrieved data. Next, it constructs a task-specific relational graph to incorporate the inter-table structure via *dynamic relation modeling*, producing enriched relation embeddings. Finally, it integrates these embeddings via *dynamic model fusion* into a unified fused embedding for the final *task-specific prediction*. This modeling process is conditioned on the query, where the task profile determines the model's structural composition, and the data profile provides the context to parametrically adjust computation, thereby establishing a flexible base model architecture capable of adaptively augmenting the base model for various analytical tasks.

In summary, we make the following contributions:

- We present NeurIDA, an autonomous system for end-to-end in-database analytics within RDBMS, which enables the automatic execution of analytical tasks without domain expertise.
- We establish a composable model architecture, a modular framework that dynamically constructs bespoke models at query time by assembling base models and shared modules, as guided by the task and data profile.
- We introduce the dynamic in-database modeling paradigm that shifts from static models to runtime model construction, reconciling the rigidity of ML with the dynamism of RDBMS.
- We evaluate NeurIDA on five real-world relational databases across ten analytical tasks. Extensive experiments show that NeurIDA achieves up to 12% improvement in AUC-ROC on classification tasks and a 10%–25% reduction in MAE on regression tasks with respect to standalone base models.

NeurIDA is a component of NeurDB [35, 61]. It has been integrated into NeurDB as its native in-database analytics engine.

The remainder of this paper is structured as follows: Section 2 introduces preliminaries. Section 3 presents the design of NeurIDA, detailing its four key modules and the dynamic model construction technique. Section 5 reports experimental results, and Section 6 reviews related work. Finally, Section 7 concludes the paper.

2 PRELIMINARIES

This section formally defines the foundational concepts for dynamic in-database modeling, It first details the structure of relational data and its schema, and then introduces the notion of a data slice.

Relational Data. A relational database $\mathcal{D}$ is a structured collection of data organized into K tables, denoted as $\mathcal{D} := \{\mathcal{T}_k\}_{k=1}^{K}$. Each table $\mathcal{T}_k$ represents a specific entity type (e.g., users, products) and consists of a set of N_k rows (tuples) and M_k columns (attributes). A row $\mathcal{T}_{k,i:}$ represents a single data instance, denoted as the vector $\mathbf{x}_i = (x_1, x_2, \ldots, x_{M^k})$ by assuming a canonical attribute ordering. These attribute values can be of heterogeneous types, e.g., numerical, categorical, text, or timestamp. A column $\mathcal{T}_{k,:j}$ corresponds to a specific attribute, holding values of a homogeneous data type for all tuples in the table. Tables are interconnected through Primary Key (PK) and Foreign Key (FK) constraints, which enforce relational integrity and specify the structural dependencies among tables.

Data Slice. Given an analytical query q, the data slice $\mathcal{D}_q$ is defined as the task-specific subset of the relational database relevant to this query, effectively forming a smaller but self-contained database derived from $\mathcal{D}$. For instance, in a user-churn prediction task, the data slice would consist of only the tables and attributes related to user profiles and past purchases. Formally, a data slice is a collection of table slices, defined as $\mathcal{D}_q = \{\mathcal{T}_{k,q} | \mathcal{T}_{k,q} \in \mathcal{D}, k \in \mathcal{K}_q \subseteq \{1, \cdots, K\}\}$, where a table slice $\mathcal{T}_{k,q}$ is derived from its corresponding source table $\mathcal{T}_k \in \mathcal{D}$ by selecting a subset of its rows and columns. This operation can be expressed as: $\mathcal{T}_{k,q} = \pi_{J_{k,q}}(\sigma_{I_{k,q}}(\mathcal{T}_k))$, where σ is the selection operator for rows with indices $I_{k,q}$, and π is the projection operator for columns with indices $J_{k,q}$, with both index sets determined by the SQL generated from query q. The data slice $\mathcal{D}_q$ thus serves as the central object for our analytics pipeline, with all subsequent modeling performed exclusively on this focused subset.

3 NEURIDA OVERVIEW

NeurIDA is designed to extend the capabilities of a traditional RDBMS to support dynamic in-database analytics. It accepts a natural language query (NLQ) as input and automatically generates a comprehensive analytical report as output. The workflow of NeurIDA, as illustrated in Figure 1, is orchestrated by four key components: (1) First, the **Query Intent Analyzer** parses the NLQ to formalize the task profile and the data profile, which specify the prediction task and required data slice, respectively. (2) Next, the **Conditional Model Dispatcher** evaluates the task complexity to select the best base model, and decides whether to directly deploy the base model or invoke advanced model augmentation. (3) Subsequently, **Dynamic In-Database Modeling Engine** (DIME) retrieves the data slice, and executes the modeling based on the decision, either directly running the base model or dynamically constructing a bespoke model for the task. (4) Finally, the **Analytical Report Synthesizer** synthesizes the prediction results within the context of the task to produce an analytical report. We shall detail each component in subsequent sections.

3.1 Query Intent Analyzer

Upon the arrival of an NLQ, the **Task Parser** and **Data Profiler** sequentially generate two structured outputs: a *task profile* and a *data profile*. The former defines the analytical task for subsequent modeling, while the latter specifies the data slice required for analytics.

Task Parser. The Task Parser takes the NLQ and the database catalog (DB catalog) as input, which describes the schema, including tables and relationships. It formulates the *task profile* implied in the query by identifying four key fields: the target table, task type, prediction window, and target query. Specifically, the target table denotes the primary entity table inferred from the NLQ and grounded in the database schema. The task type specifies the prediction objective, such as classification or regression. The prediction window defines the temporal horizon of the task, while the target query extracts the labeled data from the database for model training and evaluation. In NeurIDA, this extraction process is automated by an analytical agent powered by a Large Language Model (LLM) [50] using a carefully crafted prompt, as illustrated in Figure 2. The prompt provides structured guidance by incorporating both the

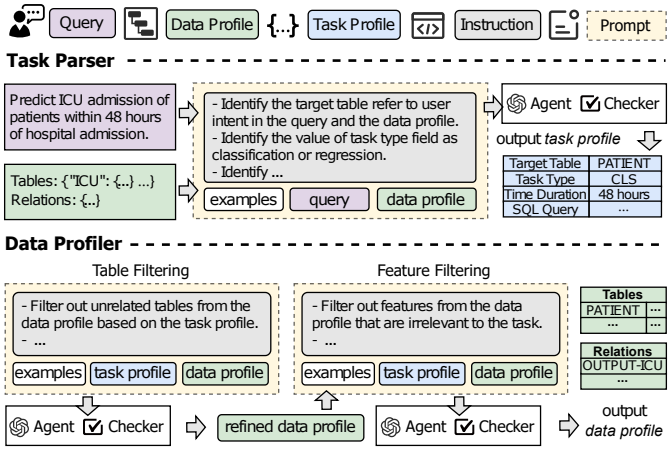


Figure 2: Workflow of the Query Intent Analyzer, illustrating how an NLQ is parsed into structured task and data profiles.

NLQ and the DB catalog as context. To ensure output consistency, the prompt includes illustrative examples demonstrating the desired response format. The agent's response is parsed into the task profile and subsequently validated by a rule checker to guarantee syntactic correctness and schema alignment.

Data Profiler. Given the task profile, the Data Profiler constructs a data profile for subsequent modeling, which identifies the task-relevant subset of tables, relationships, and columns. In essence, the Data Profiler acts as a domain expert that curates the optimal schema context for the predictive task. To automate this complex reasoning, NeurIDA again employs an LLM-based analytical agent. A chain-of-thought (CoT) prompting strategy is adopted to decompose the profiling task into multiple rounds of interactions, as illustrated at the bottom of Figure 2. The agent first identifies the related tables connected to the target table defined in the *task profile*. Then, it refines the selected tables by filtering out irrelevant columns, such as internal database metadata or attributes lacking predictive signal. The resulting data profile is integrated with the task profile to drive the subsequent modeling operations.

3.2 Conditional Model Dispatcher

Before invoking the DIME engine, NeurIDA employs a lightweight **Conditional Model Dispatcher** to optimize efficiency for data analytics. This component addresses two problems: (1) Base Model Selection, determining which pre-trained base model m_i from the model pool $\mathcal{M} = \{m_1, m_2, \ldots, m_K\}$ is best suited for the current analytical task; and (2) Conditional Augmentation, deciding whether the base model m_i provides sufficient performance, or requires the relational augmentation via DIME, i.e., transforming m_i into a dynamically constructed relational model, as invoking model augmentation could be unnecessary for simple tasks where a standalone base model can excel. Therefore, the dispatcher maintains a metadata dictionary $\mathcal{D}_p$ that tracks the historical performance statistics, particularly the Exponential Moving Average (EMA) of each base model m_i, denoted as μ_i. Upon receiving the *task profile* and the *data profile*, the dispatcher first constructs a small evaluation

Table 1: Notations used in the Conditional Model Dispatcher.

Notation	Description
m_i	Candidate base model.
μ_i	Historical EMA performance of m_i.
s_i	Zero-Cost Proxy (ZCP) score of m_i.
m^*	Optimal base model selected via ZCP.
τ	Selection threshold derived from the EMA of m^*.
ϵ	Performance tolerance parameter, $\epsilon \in [0, 1]$.

batch by randomly sampling labeled tuples from the data slice. It then employs Zero-Cost Proxies (ZCP) [5, 46], a training-free evaluation technique originally proposed in Neural Architecture Search (NAS) [47, 57], to efficiently estimate the predictive performance of each candidate base model without any training. ZCPs estimate model quality at initialization by analyzing gradient and activation statistics over the sampled evaluation batch. In practice, each candidate model performs a single forward pass and an optional backward pass to compute a proxy score s_i, which correlates with the model's expected full-training performance. The candidates are subsequently ranked by their proxy scores, and the one with the highest score s^*, is selected as the base model, m^*.

The dispatcher then determines whether to invoke the augmentation by comparing s^* against a dynamic threshold τ, defined as $\tau = (1 - \epsilon) \cdot \mu_{m^*}$, where μ_{m^*} is the historical EMA of m^* on the task and ϵ is a tolerance parameter controlling acceptable performance deviation. If $s^* \geq \tau$, indicating that the base model maintains its performance within an acceptable margin of its historical standard, the system bypasses structural augmentation and directly deploys m^* for prediction. Otherwise, if $s^* < \tau$, the system invokes DIME to augment m^* with relational context, dynamically constructing a bespoke model (denoted as m^* w/ NeurIDA) to bridge the performance gap. This component ensures that NeurIDA judiciously allocates computational resources, triggering augmentation only when the performance drop exceeds the tolerance threshold ϵ.

3.3 Dynamic In-database Modeling Engine

Once the modeling strategy is determined, DIME retrieves the corresponding data slice and dynamically constructs a bespoke model for the analytical task. A key challenge in such dynamic modeling over relational databases is the extreme variability of analytical tasks and their associated data profiles. Static modeling is inherently limited: a single general-purpose model is inevitably suboptimal across diverse tasks, whereas training a dedicated model from scratch for each new task is prohibitively expensive. To resolve this, DIME introduces a dynamic modeling paradigm. At its core is a modular and composable framework that comprises a pool of heterogeneous base models and shared model components. Given the task and data profile, DIME dynamically instantiates the predictive model by augmenting the selected base model with shared components. This framework establishes a composable modeling backbone that jointly captures individual-table semantics and inter-table relational structures. By design, this architecture enables adaptive modeling while maintaining transferability across diverse analytical tasks. The detailed execution flow of DIME is illustrated in Section 4.

3.4 Analytical Report Synthesizer

NeurIDA generates predictions for tuples in the target table via DIME. However, these outputs remain at an instance-level and require further interpretation to support high-level decision-making. Translating these results into actionable insights, such as the drivers of customer churn or emerging market trends, traditionally requires analysis and summarization by domain experts.

To automate this process, this component formulates the problem as an evidence-grounded generation task. Given the *task profile*, the *data slice*, and the prediction results, an analytical agent powered by LLMs examines these results and retrieves supporting evidence from data, such as representative tuples, statistical patterns, or aggregated trends. Based on this evidence, the agent reasons about the underlying signals and composes a structured analytical report aligned with the user's NLQ.

4 THE DIME MODELING FRAMEWORK

In this section, we present the DIME modeling framework and its execution flow. When model augmentation is invoked, DIME dynamically constructs a task-specific model within the composable framework. It first represents the data slice as a relational graph, where tuples from the target table and related tables are connected through Primary-Foreign Key (PK-FK) relationships. Based on this graph, DIME instantiates a bespoke model using the selected base model together with shared model components, and applies it to generate predictions for target tuples.

As illustrated in Figure 3, the modeling pipeline consists of four stages. (1) **Base Table Embedding**: generates initial tuple embeddings to capture individual-table semantics for tuples in the target table and related tables via a selected base model and a *unified tuple encoder*, respectively. (2) **Dynamic Relation Modeling**: enriches the tuple embedding with inter-table structures from the relational graph, producing relation embeddings through a shared *relation-aware message passing* module. (3) **Dynamic Model Fusion**: enhances the representation of each target tuple into a fused embedding by incorporating relational context from neighboring related tables through a *context-aware fusion* module. (4) **Task-Aware Prediction**: feeds the representation of the target tuple into a task-specific prediction head for the final prediction. In the following subsections, we describe each component and explain how a bespoke model is dynamically constructed from these model components across different databases and tasks.

Base Table Embedding. The primary objective of this stage is to capture *intra-table semantics* for all tuples in the retrieved data slice, providing the initial representation required for the subsequent relational augmentation. Given the selected base model m^*, DIME employs a dual-path embedding strategy. First, m^* is applied to its original input features to produce native representations that preserve its inherent inductive biases. In parallel, DIME introduces a shared model component, the *unified tuple encoder*, which generates representations for all tuples across heterogeneous tables. The output vectors from both the base model and the unified tuple encoder are collectively referred to as tuple embeddings $\mathbf{e}$. Specifically, the unified tuple encoder transforms tuples from respective table slices into embeddings in a common d-dimensional representation space. Given a tuple $\mathbf{x} = (x_1, x_2, \cdots, x_M)$ with M attributes in a

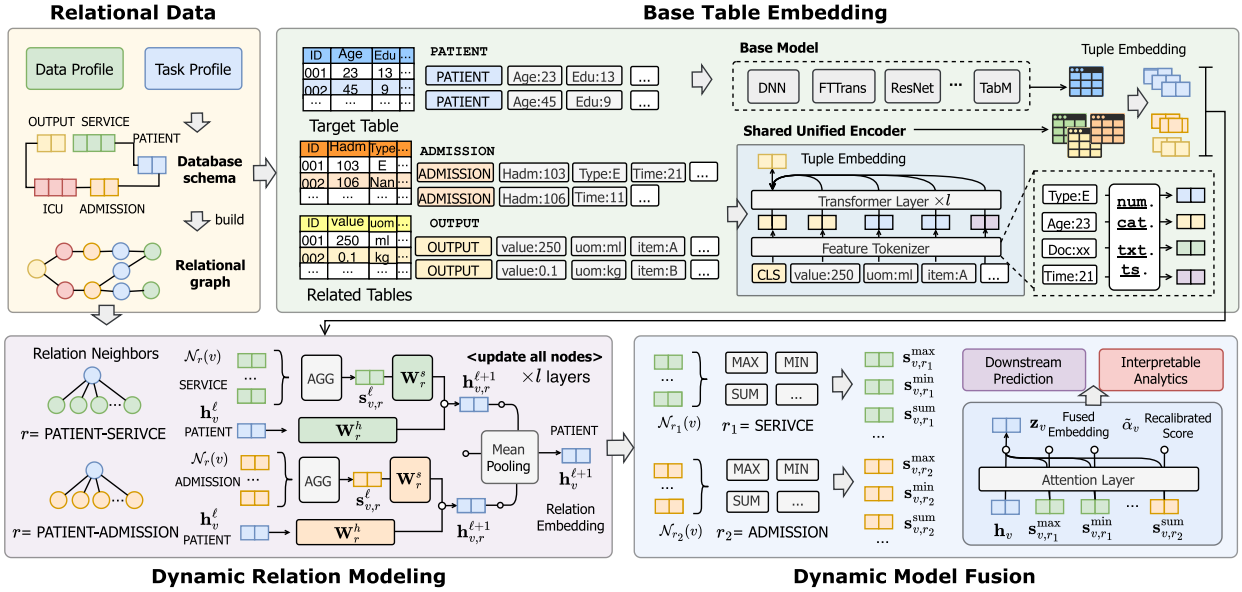


Figure 3: Overview of the Dynamic In-Database Modeling Engine.

table slice $\mathcal{T}$, the encoder first encodes each raw attribute value x_i into a dense embedding vector $\mathbf{e}_i$ corresponding to its data type.

Specifically, for categorical attributes, we use a learnable embedding lookup table, i.e., $\mathbf{e}_i = \mathbf{E}_i[x_i]$. For numerical attributes, we apply a linear transformation with learnable parameters, i.e., $\mathbf{e}_i = x_i \cdot \hat{\mathbf{e}}_i + \mathbf{b}_i \quad \hat{\mathbf{e}}_i$ where $\mathbf{b}_i \in \mathbb{R}^d$. For textual attributes, we concatenate the column name with the attribute value, encode the resulting text using a shared pre-trained language model $\mathrm{LM}(\cdot)$, and then project it to dimension d via a linear layer, i.e., $\mathbf{e}_i = \mathrm{Linear}(\mathrm{LM}([\texttt{col\_name}, x_i]))$. For timestamp attributes, we use a Time2Vec [52] encoder to capture absolute progression and periodic patterns. Concretely, each timestamp is decomposed into an absolute part t_{year} and a cyclical part $\mathbf{t}_{\text{cyc}} = [t_{\text{month}}, t_{\text{day}}, t_{\text{hour}}]$, and encoded as $\mathbf{e}_t = [\phi_{\text{abs}}(t_{\text{year}} - t_{\text{base}}), \phi_{\text{cyc}}(\mathbf{t}_{\text{cyc}})]\mathbf{W}_t + \mathbf{b}_t$, where ϕ_{abs} is a learnable scaling function, ϕ_{cyc} uses sinusoidal mappings for periodicity. In this way, all heterogeneous attributes are projected into a unified latent space $\mathbb{R}^d$.

Next, the set of attribute embeddings $\{\mathbf{e}_1, \ldots, \mathbf{e}_M\}$ is then processed by a transformer-based module consisting of a Multi-Head Self-Attention (MSA) layer and a Feed-Forward Network (FFN). The self-attention mechanism models attribute interactions to capture higher-order dependencies, and its permutation-invariance naturally aligns with the unordered nature of attributes in a relational tuple. A learnable table-specific `[CLS]` token $\mathbf{e}_{\text{cls}}$ is prepended for summarization. Let $\mathbf{E} = [\mathbf{e}_{\text{cls}}, \mathbf{e}_1, \ldots, \mathbf{e}_M] \in \mathbb{R}^{(M+1)\times d}$ denote the input sequence. For attention head $h \in \{1, \ldots, H\}$:

$$\mathrm{Attn}_h(\mathbf{E}) = \mathrm{softmax}\left(\frac{(\mathbf{E}\mathbf{W}_Q^{(h)})(\mathbf{E}\mathbf{W}_K^{(h)})^{\mathrm{T}}}{\sqrt{d_h}}\right)(\mathbf{E}\mathbf{W}_V^{(h)})$$

where $\mathbf{W}_Q^{(h)}, \mathbf{W}_K^{(h)}, \mathbf{W}_V^{(h)} \in \mathbb{R}^{d\times d_h}$ are projection matrices. The outputs are concatenated and projected back to dimension d:

$$\mathrm{MSA}(\mathbf{E}) = \mathrm{Concat}(\mathrm{Attn}_1(\mathbf{E}), \cdots, \mathrm{Attn}_H(\mathbf{E}))\mathbf{W}_O$$

where $\mathbf{W}_O \in \mathbb{R}^{H\cdot d_h \times d}$ is the output projection. Residual connections and layer normalization are applied to stabilize the embedding process: $\mathbf{H}' = \mathrm{LayerNorm}(\mathbf{E} + \mathrm{MSA}(\mathbf{E}))$. The embeddings are then refined by a subsequent FFN layer: $\mathbf{H} = \mathrm{LayerNorm}(\mathbf{H}' + \mathrm{FFN}(\mathbf{H}'))$. Finally, the tuple embedding $\mathbf{c}$ is obtained by mean-pooling the output sequence $\mathbf{H}$, followed by layer normalization:

$$\mathbf{c} = \mathrm{LayerNorm}(\mathrm{MEAN}(\mathbf{H}))$$

This representation captures fine-grained intra-table information through aggregated attribute interactions learned by the encoder. By sharing this model component across diverse table slices, DIME establishes a highly scalable and unified interface for augmenting the base model in subsequent relational modeling.

Crucially, this design facilitates the seamless augmentation of any selected base model, e.g., FT-Transformer [19] or ARM-Net [10]. By generating tuple embeddings via the unified tuple encoder for the supporting table slices, DIME effectively "wraps" the base model with rich inter-table structural context in subsequent stages. The mechanism dynamically extends the capabilities of the base model, enabling it to exploit the full relational schema without altering its internal feature processing logic.

Dynamic Relation Modeling. In this stage, the given tuple embeddings are refined by integrating *inter-table structures*. Specifically, the task-specific relational graph is explicitly modeled to capture structural interactions within tuples defined by the underlying schema constraints.

Formally, the data slice is represented as a heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{A})$, where $\mathcal{A}$ denotes node types (tables, e.g., `USER`, `ITEM`), and $\mathcal{R}$ represents the set of relation types (PK-FK links, e.g., `USER-PURCHASE-ITEM`). Each node $v \in \mathcal{V}$ corresponds to a tuple, and each edge $(u, r, v) \in \mathcal{E}$ represents an undirected PK-FK link of relation type $r \in \mathcal{R}$. The relation-specific neighborhood $\mathcal{N}_r(v)$ is defined as the set of nodes u connected to v via the relation type

r: $\mathcal{N}_r(v) = \{u|(u, r, v) \in \mathcal{E}\}$. For each node v, the representation at layer $\ell = 0$ is initialized as $\mathbf{h}_v^{(\ell)} = \mathbf{c}_v$ (the tuple embedding from the previous stage). A shared *relation-aware message passing* module then propagates information along the relational graph. For each relation type r, neighbor messages are first aggregated:

$$\mathbf{s}_{v,r}^{(\ell)} = \text{AGG}(\mathbf{h}_u^{(\ell)} : u \in \mathcal{N}_r(v))$$

where $\text{AGG}(\cdot)$ is an aggregate function (e.g., mean or sum) that produces a relation-specific local summary $\mathbf{s}_{v,r}^{(\ell)}$. It is then integrated into the current node representation:

$$\mathbf{h}_{v,r}^{(\ell+1)} = \mathbf{W}_r^h \mathbf{h}_{v,r}^{(\ell)} + \mathbf{W}_r^s \mathbf{s}_{v,r}^{(\ell)}$$

where $\mathbf{W}_r^h, \mathbf{W}_r^s \in \mathbb{R}^{d \times d}$ are relation-specific learnable parameters. In this design, the model captures different semantic roles across schema relations. For instance, `item–category` links reflect hierarchical structure, whereas `user–item` links represent interaction patterns. Maintaining relation-specific parameters enables the model to adaptively interpret neighbor information from highly diverse schema contexts. Since a node may participate in multiple relations (e.g., a product linked to a category, an order, and a user review), its final representation $\mathbf{h}_v$ (relational embedding) consolidates messages across all relation types via a lightweight cross-relation aggregation with normalization and nonlinearity:

$$\mathbf{h}_v^{(\ell+1)} = \text{LayerNorm}(\phi(\frac{1}{|\mathcal{R}_v|} \sum_{r \in \mathcal{R}_v} \mathbf{h}_{v,r}^{(\ell+1)}))$$

where $\phi(\cdot)$ is the activation function, and $\mathcal{R}_v$ denotes the set of relation types associated with node v. Notably, the resulting relational embedding remains instance-dependent: each node aggregates messages only from the active relation set and sampled context neighbors, reflecting the dynamically induced relational structure.

Overall, this shared model component enables tuple embeddings to incorporate rich structural information from the database schema. By stacking multiple layers of this model component, the receptive field of each relational embedding naturally expands to encompass multi-hop neighborhoods. This allows the constructed model to encode higher-order interactions across the specified relational graph, thereby capturing inter-table dependencies beyond the immediate connections of individual tables.

Dynamic Model Fusion. While the preceding dynamic relation modeling stage implicitly enriches tuple representations through structural message passing, this stage explicitly synthesizes a task-specific summary from the relational neighborhood of the target tuple for final prediction. At this point, the relational embedding $\mathbf{h}_v$ already incorporates both intra-table semantic and inter-table structural information. However, maximizing predictive performance requires distinguishing which specific relational statistics, e.g., the *maximum* purchase versus the *average* rating in a repurchase task, are more discriminative for the task at hand.

To this end, we introduce a *context-aware fusion* module, which adaptively fuses the target tuple embedding with distinct aggregated relational contexts. Specifically, for each target tuple v, a set of relation-specific contextual signals is generated via multiple aggregation functions to the neighbors $\mathcal{N}_r(v)$.

$$\mathbf{s}_{v,r}^{AGG} = \text{AGG}(\{\mathbf{h}_u : u \in \mathcal{N}_r(v)\})$$

where $\text{AGG}(\cdot) \in \mathcal{F}$ denotes a set of adopted aggregate functions (Max, Min, Sum, Mean). Each resulting signal $\mathbf{s}_{v,r}^{\text{AGG}}$ captures a distinct statistical property of neighbors connected via relation type r. These signals are organized into a contextual sequence $\mathcal{S}_v$, which concatenates the target embedding $\mathbf{h}_v$ with all derived contextual signals $\mathbf{s}_{v,r}^{\text{AGG}}$ across the associated relation types $r \in \mathcal{R}_v$:

$$\mathcal{S}_v = [\mathbf{h}_v, \mathbf{s}_{v,r_1}^{\max}, \ldots, \mathbf{s}_{v,r_1}^{\text{sum}}, \ldots, \mathbf{s}_{v,r_{|\mathcal{R}_v|}}^{\max}, \ldots, \mathbf{s}_{v,r_{|\mathcal{R}_v|}}^{\text{sum}}]$$

This sequence is then processed by a task-specific Multi-Head Self-Attention layer to capture interactions among these contextual signals. The final fused embedding $\mathbf{z}_v$ is obtained by retrieving the output vector corresponding to the target tuple embedding $\mathbf{h}_v$:

$$\mathbf{z}_v = \text{MSA}(\mathcal{S}_v)[0]$$

This module complements the previous stage by acting as a fine-grained summarizer. It is trained end-to-end for the target task, enabling selective attention over informative relational signals. The resulting fused embedding provides a more robust and task-specific representation for final prediction.

Interpretability. Beyond improving prediction performance, the self-attention mechanism inherently supports model interpretability. The attention weights generated during fusion form a probability distribution over the contextual sequence $\mathcal{S}_v$, indicating the relative contribution to the final fused embedding from different aggregated relational statistics $\mathbf{s}_{v,r}^{AGG}$, such as payment sums or average ratings. By analyzing these weights, DIME offers transparency into the dynamically constructed model, revealing which specific relational signals are more influential for the given prediction task.

However, raw attention weights are not directly comparable across tuples, since the length of the contextual sequence $|\mathcal{S}_v|$ varies with the number of associated relation types. For instance, the same weight may indicate high importance in a long sequence but only moderate importance in a short one. To address this, DIME computes a recalibrated importance score $\tilde{\alpha}_{v,i}$ by quantifying the deviation of the raw weight $\alpha_{v,i}$ from the uninformative uniform baseline $b = 1/|\mathcal{S}_v|$:

$$\tilde{\alpha}_{v,i} = \max\left(\frac{\alpha_{v,i} - b}{1 - b}, 0\right), i \in \{0, 1, \ldots, |\mathcal{S}_v| - 1\}.$$

This normalization provides a length-independent metric, facilitating comparable interpretability of importance across diverse relational signals.

Task-Aware Prediction. Finally, the fused embedding $\mathbf{z}_v$ of the target tuple passes through an activation and normalization layer, serving as the input for a task-specific prediction head $g_\theta(\cdot)$ for final prediction:

$$\hat{y} = g_\theta(\text{LayerNorm}(\phi(\mathbf{z}_v))).$$

Here, the prediction head $g_\theta(\cdot)$ and the output $\hat{y}$ are determined by the *task profile*. For instance, for a classification task, $\hat{y}$ represents the predicted class probability, while for a regression task, it corresponds to a continuous numerical score.

Table 2: Statistics of databases and prediction tasks.

Dataset	#Table	#Relation	#Column	Domain	Task	Type	Target Table #Instance		#Attribute		#Instance		
									Target*	All†	Train	Valid	Test
Event	5	7	117	Sociality	user-repeat	CLS	users	37,143	8	25	3,842	268	246
					user-attend	REG	users	37,143	8	19	19,239	2,013	1,958
Beer	9	12	116	E-commerce	user-active	CLS	users	89,662	11	47	16,656	2,794	3,558
					beer-score	REG	beers	719,164	21	43	45,922	12,858	7,218
Trial	15	15	77	Healthcare	study-result	CLS	studies	249,730	27	50	11,994	960	825
					site-success	REG	facilities	453,233	7	14	100,000	19,740	22,617
Avito	8	11	26	Online Ads	user-clicks	CLS	user_info	98,250	6	15	59,454	21,183	47,996
					ad-ctr	REG	ad_info	5,960,558	8	18	5,100	1,766	1,816
HM	3	2	37	Retail	user-churn	CLS	customer	1,371,980	8	13	100,000	76,556	74,575
					item-sales	REG	article	105,542	26	32	100,000	100,000	100,000

∗ (#Attribute) Target: denotes the number of attributes available within the target table's schema.
† (#Attribute) All: represents the total cumulative number of unique attributes across the entire relational database (the union of all table schemas).

Optimization. To balance model transferability and adaptability across diverse analytical tasks, DIME employs a hybrid optimization strategy. Foundational modules, i.e., the *unified tuple encoder* and the *relation-aware message passing* module, are pre-trained and shared across tasks to capture universal database patterns. In contrast, task-specific model components, including the *context-aware fusion* module, the prediction head, and any integrated advanced *base model*, e.g., FT-Transformer [19] or ARM-Net [10], are optimized or fine-tuned in a task-specific manner to maximize performance for the current analytical task. For binary classification tasks, the optimization function is the binary cross-entropy loss:

$$\mathcal{L}(\hat{y}, y) = -\frac{1}{N}\sum_{i=1}^{N}\{y_i \log\sigma(\hat{y}_i) + (1-y_i)\log(1-\sigma(\hat{y}_i))\}$$

For regression tasks, the optimization function employs the L1 loss:

$$\mathcal{L}(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$$

where y denotes the ground truth label and N is the number of training instances in the *data slice*.

5 EXPERIMENTS

In this section, we evaluate NeurIDA through experiments on multiple real-world relational databases and various prediction tasks. We further analyze its modules through ablation and sensitivity studies, examine system cost, and quantify the agent-based components.

5.1 Experimental Setup

Datasets and Tasks. We conduct studies on five relational databases from healthcare, sociality, and e-commerce domains. Table 2 provides the statistics of databases and associated prediction tasks.

• **Event** [1] is a recommendation database from a mobile social-planning app, containing users, invitations, and event information. There are two tasks: (i) predicting whether a user will re-attend a joined event (user-repeat), and (ii) estimating how many invitations the user will respond yes to in the next week (user-attend).

• **Beer** [29] is from a beer-review platform that records users, beers, and reviews information. It involves two tasks: (i) predicting whether a user will write more than ten reviews in the next season (user-active), and (ii) predicting the positive-rating ratio, where reviews with scores above 3.5 are treated as positive (beer-score).

• **Trial** [4] is a clinical trial database , containing studies, designs, and outcomes. We evaluate two tasks for the next year: (i) predicting whether a clinical study will be successful (study-result), and (ii) estimating the site-level success rate (site-success).

• **Avito** [3] is a database of an online advertising platform that records user interactions such as searches, visits, and phone contacts. We consider two tasks for the next four days: (i) predicting whether a user will click on more than one advertisement (user-clicks), and (ii) estimating the click-through rate (ad-ctr).

• **HM** [2] is a retail database containing customer profiles, item information, and transaction records. We define two tasks for the next week: (i) predicting whether a customer will churn, i.e., make no transaction (user-churn), and (ii) predicting the item-level sales volume, measured by total transaction value (item-sales).

Baselines. We use a variety of base models as baselines, which are organized into the following four groups:

• **Standalone Tabular Models (STM).** This group operates on target tables and serves as a strong baseline for tabular analytics. It includes classical machine learning models such as Logistic Regression (LR) [13] and Random Forest (RF) [8], advanced tree-based models such as CatBoost [43] and LightGBM [25], and recent tabular foundation models, including TabPFN [22] and TabICL [44], which adapt to new prediction tasks without training.

• **Tuple Representation Models (TRM).** These neural tabular models learn tuple representations that can be further enhanced by NeurIDA with relational structure. We include a diverse set of architectures in the model pool: DNN, DeepFM [20], ResNet [19], FT-Transformer (FTTrans) [19], ARM-Net [10], and TabM [18].

• **AutoML.** To ensure competitive model configurations, we integrate automated learning approaches, including Optuna [6] for hyperparameter optimization and Trails [56] for model architecture search. These approaches operate orthogonally to NeurIDA.

• **Large Tabular Models (LTM).** We further evaluate pre-trained language models that treat each tuple as text and encode it into dense representations. LTM includes TP-BERTa [58], a RoBERTa-based model for tabular analytics tasks, as well as Nomic [34] and BGE [55], two general text embedding models pre-trained on large-scale corpora for semantic representation learning.

Table 3: Overall effectiveness of NeurIDA across five relational databases.

Task Type (Metric)		Classification (AUC-ROC) ↑					Regression (MAE) ↓				
Dataset /Task		Event	Beer	Trial	Avito	HM	Event	Beer	Trial	Avito	HM
Type	Method	user-repeat	user-active	study-result	user-clicks	user-churn	user-attend	beer-score	site-success	ad-ctr	item-sales
STM	LR	0.7376	0.8812	0.6881	0.6407	0.6182	0.3912	0.2046	0.4594	0.0483	0.0659
	RF	0.7270	0.8737	0.6770	0.6450	0.6104	0.3745	0.1997	0.4565	0.0485	0.0584
	CatBoost	0.7429	0.9060	0.6945	0.6504	0.6232	0.2607	0.1975	0.4429	0.0393	0.0557
	LightGBM	0.7340	0.9061	0.6999	0.6527	0.6241	0.2547	0.1903	0.4595	0.0379	0.0495
	TabPFN	0.7754	0.9164	0.7051	0.6437	0.6254	0.3406	0.1903	0.4639	0.0369	0.0583
	TabICL*	0.7692	0.9077	0.7018	0.6503	0.6371	-	-	-	-	-
TRM	DNN	0.7294	0.9064	0.6830	0.6546	0.6290	0.2523	0.1947	0.4500	0.0394	0.0551
	↪w/ NeurIDA	**0.7901**	**0.9289**	**0.7057**	**0.6714**	**0.6837**	**0.2403**	**0.1719**	**0.3919**	**0.0366**	**0.0425**
	DeepFM	0.7130	0.9047	0.7013	0.6283	0.6203	0.2491	0.2091	0.4581	0.0397	0.0541
	↪w/ NeurIDA	**0.7993**	**0.9366**	**0.7042**	**0.6737**	**0.6833**	**0.2469**	**0.1727**	**0.4077**	**0.0365**	**0.0419**
	ResNet	0.7461	0.9055	0.7044	0.6587	0.6332	0.2422	0.1891	0.4441	0.0378	0.0599
	↪w/ NeurIDA	**0.7973**	**0.9379**	**0.7097**	**0.6740**	**0.6874**	**0.2376**	**0.1715**	**0.3713**	**0.0345**	**0.0390**
	FTTrans	0.7346	0.9131	0.6836	0.6502	0.6304	0.2539	0.1825	0.4290	0.0374	0.0584
	↪w/ NeurIDA	**0.8012**	**0.9360**	**0.7171**	**0.6730**	**0.6655**	**0.2401**	**0.1696**	**0.3817**	**0.0347**	**0.0398**
	ARM-Net	0.7402	0.9016	0.6965	0.6604	0.6297	0.2642	0.1912	0.4468	0.0368	0.0515
	↪w/ NeurIDA	**0.7934**	**0.9394**	**0.7130**	**0.6783**	**0.6900**	**0.2372**	**0.1673**	**0.3906**	**0.0349**	**0.0403**
	TabM	0.7415	0.9014	0.7048	0.6610	0.6270	0.2505	0.1829	0.4087	0.0371	0.0510
	↪w/ NeurIDA	**0.8028**	**0.9370**	**0.7166**	**0.6891**	**0.6837**	**0.2397**	**0.1692**	**0.3812**	**0.0352**	**0.0408**
AutoML	Optuna	0.7538	0.9112	0.7021	0.6392	0.6287	0.2378	0.1841	0.4069	0.0389	0.0533
	↪w/ NeurIDA	**0.7993**	**0.9317**	**0.7192**	**0.6755**	**0.6793**	**0.2335**	**0.1703**	**0.3891**	**0.0345**	**0.0401**
	Trials	0.7602	0.9130	0.7040	0.6568	0.6225	0.2480	0.1919	0.4023	0.0401	0.0497
	↪w/ NeurIDA	**0.8017**	**0.9301**	**0.7142**	**0.6657**	**0.6739**	**0.2391**	**0.1710**	**0.3745**	**0.0373**	**0.0399**
LTM	TP-BERTa†	0.5457	0.5170	-	0.5122	0.5058	0.2768	0.3155	0.4612	0.0430	0.3514
	↪w/ NeurIDA	**0.7705**	**0.9248**	-	**0.6472**	**0.6677**	**0.2518**	**0.1864**	**0.4281**	**0.0403**	**0.0455**
	Nomic	0.6896	0.8896	0.6533	0.5771	0.5313	0.2677	0.3439	0.4545	0.0435	0.2063
	↪w/ NeurIDA	**0.7730**	**0.9178**	**0.6760**	**0.6558**	**0.6663**	**0.2504**	**0.1842**	**0.4228**	**0.0397**	**0.0453**
	BGE	0.6787	0.8868	0.6503	0.6462	0.6097	0.2645	0.2829	0.4511	0.0410	0.0772
	↪w/ NeurIDA	**0.7857**	**0.9224**	**0.6805**	**0.6580**	**0.6657**	**0.2474**	**0.1938**	**0.4098**	**0.0401**	**0.0481**

∗ TabICL: designed for classification, incompatible with regression tasks.
† TP-BERTa: the sequence length required by the Trial (study-result) task exceeds the TP-BERTa maximum input limit (512 tokens).

Evaluation. In the main study, we report *the area under the curve* (AUC-ROC) for classification tasks, where a higher value indicates better discriminative performance. We use the *mean absolute error* (MAE) for regression tasks, where a lower value denotes higher prediction accuracy. To ensure temporal consistency and prevent information leakage, each dataset is chronologically partitioned into training, validation, and test sets based on global time cutoffs. The specific cutoff points vary across prediction tasks and databases.

To ensure fair and robust evaluation, all results are averaged over three independent runs. We apply *early stopping* [42] on the validation set. Specifically, training stops if the validation metric does not improve for a predefined number of consecutive epochs, and the best-performing checkpoint is retained for testing.

Implementation. NeurIDA adopts a hybrid implementation strategy that balances in-database efficiency with LLM flexibility. Specifically, the performance-critical modeling components, namely the dispatcher and DIME, are implemented as user-defined functions (UDFs). This native execution minimizes data movement and accelerates inference. In contrast, the analyzer and synthesizer operate as external services to simplify LLM integration and management, as LLMs typically require substantial memory and GPU resources.

For the evaluation of DIME, all approaches are configured consistently. In particular, FTTrans uses 4 attention heads, ARM-Net uses 1 attention head, and TabM employs 8 ensemble experts to balance efficiency and accuracy. For the AutoML approaches, ResNet serves as the backbone model, with the search space defined by the number of layers and hidden widths with candidate values in $32, 64, 128, \ldots, 512$. Both Trails and Optuna automatically explore the space to select the best architecture for each task. For the LTM group, we treat each pre-trained model as a frozen encoder that maps a tuple into a semantic embedding. We use the public multitask checkpoint for TP-BERTa, and nomic-embed-text-v1.5 and bge-base-en-v1.5 for Nomic and BGE, respectively. All three models produce 768-dimensional embeddings, ensuring consistent representation sizes across LTM baselines.

In model training, we use the AdamW [28] with a learning rate searched within $[10^{-3}, 10^{-1}]$ and a batch size of 256. The patience epoch in *early stopping* is set to 10 for regression and 5 for classification. All experiments are conducted on a server with a Xeon Silver 4114 CPU @ 2.2GHz (10 cores), 256GB of memory, and 8 GeForce RTX 3090 Ti. Model implementations are based on PyTorch 2.1.0 [40], Pytorch Geometric 2.5.3 [15], with CUDA 11.8.

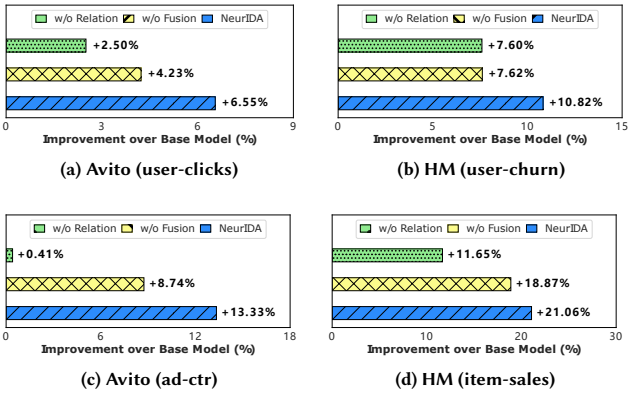


Figure 4: Ablation study for *Dynamic Relation Modeling* (w/o Relation) and *Dynamic Model Fusion* (w/o Fusion).

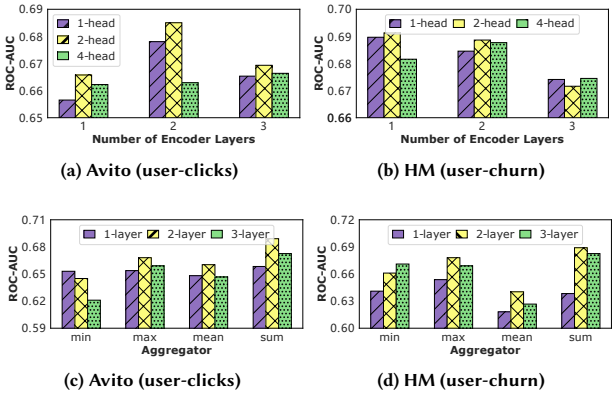


Figure 5: Effects of encoder layer l and attention head H (a-b), aggregator agg and message-passing depth ℓ (c-d).

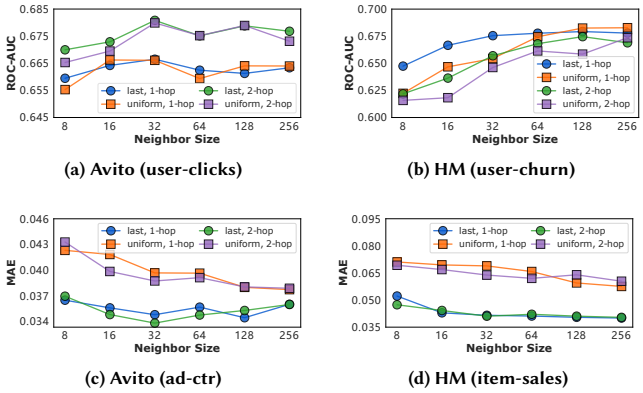
5.2 Overall Performance Evaluation

The main result is presented in Table 3, where we evaluate the performance of base models across different prediction tasks. For models that support tuple representation learning, we further compare their performance with and without the augmentation of NeurIDA. Based on these results, we draw the following key observations:

Consistent Performance Augmentation with NeurIDA. Across all prediction tasks, NeurIDA consistently improves the performance of different base models. For classification, it brings relative gains of 4%–12% in AUC-ROC, while it reduces MAE by 10%–25% relative to the standalone base model. The improvements are especially pronounced for LTM approaches. For example, on the user-repeat task, augmenting TP-BERTa with NeurIDA improves AUC-ROC from 0.55 to 0.77. Moreover, the enhanced model consistently outperforms strong standalone tabular models (STM) such as TabPFN and TabICL. These gains mainly come from the dynamic modeling design in DIME. The *unified tuple encoder* captures intra-table semantics and feature interactions within related tables, while the *relation-aware message passing* module models inter-table dependencies across relations. The *context-aware fusion* module further emphasizes task-relevant relational context for prediction. In contrast, base models without augmentation treat tuples independently and cannot exploit cross-table structure. By jointly modeling intra-table semantics and inter-table dependencies, NeurIDA generates richer tuple representations and boosts predictive performance.

Variability Across Model Groups. The performance of base models varies substantially across model groups. In STM, advanced tree-based models such as CatBoost outperform classical ML baselines like LR and RF, but do not show clear advantages over models in TRM, because high-cardinality categorical values are difficult for these models to exploit effectively without feature engineering. By comparison, tabular foundation models, including TabPFN and TabICL, provide the strongest baselines in this group, benefiting from pre-trained priors and in-context adaptation. TRMs show consistently strong expressiveness. Models such as TabM and ARM-Net achieve robust results, while simpler models like DNN perform worse in comparison. This suggests the importance of model architectural design, as attention mechanisms and expert-ensemble structures are better suited to capture complex feature interactions. AutoML methods also achieve competitive performance by searching over large hyperparameter and architecture spaces, often approaching the best model in TRMs, although their effectiveness depends on the available computation budget. By contrast, LTMs perform worse than other groups, revealing the limitations of language-model-based encoders for tabular analytics. One important reason is that many relational attributes, such as `UserAgentID` and `UserDeviceID` in Avito, are identifier-like tokens rather than semantically meaningful text, making it difficult for language models to learn informative tuple representations. Moreover, these pre-trained language models are frozen during downstream training and therefore cannot adapt to specific tasks, which further limits the capability of LTMs. Overall, the results show that differences in architecture, feature handling, and attribute semantics strongly affect the effectiveness of base models.

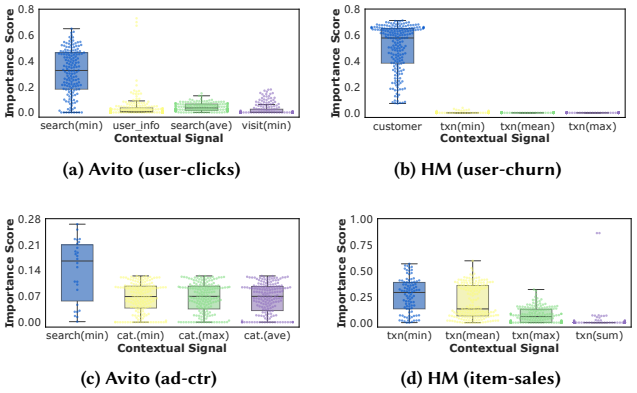
Variability in Prediction Tasks. The augmentation brought by NeurIDA varies across different prediction tasks. At a macro level, the performance gains in regression tasks are generally more substantial than in classification tasks. Regression requires precise numerical estimation and is therefore more sensitive to the quality of representation and modeling. The improvement achieved by NeurIDA in regression highlights the necessity of fine-grained dynamic modeling when the prediction objective becomes challenging. By jointly leveraging the contextual information, NeurIDA enriches tuple representations with comprehensive signals, enabling more accurate numerical prediction in regression tasks. Additionally, the gains in classification also differ across datasets. For example, the improvement on user-repeat of the Event database is much larger than that on the study-result of the Trial database. This discrepancy reflects how much the task depends on relational structure. In user-repeat, the target table (8 features) contains limited behavior information, so most predictive signals come from related tables, making NeurIDA particularly effective. In contrast, the study-result relies more on attributes already available in the target table (27 features), so the relative benefit of NeurIDA is smaller.



(a) Avito (user-clicks) (b) HM (user-churn)

(c) Avito (ad-ctr) (d) HM (item-sales)

Figure 6: Effects of neighbor sampling configuration (including neighbor size, hop number, and sampling strategy).



(a) Avito (user-clicks) (b) HM (user-churn)

(c) Avito (ad-ctr) (d) HM (item-sales)

Figure 7: Interpretability visualization and analysis based on context-aware fusion module.

5.3 Ablation and Parameter Analysis

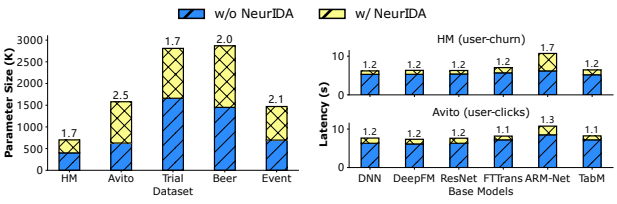
Ablation Study. To analyze the contribution of different stages in DIME, we ablate the *relation-aware message passing* module and the *context-aware fusion* module, while retaining the *unified tuple encoder* for basic tuple encoding. The two variants are denoted as w/o Relation and w/o Fusion, respectively. We evaluate their impact on the performance gain over the base model ResNet. As Figure 4 shows, removing either module degrades performance, indicating that both contribute to the effectiveness of NeurIDA. Among them, the *relation-aware message passing* module has a larger impact: without it, the performance gain drops from 6.5% to 2.5% on user-clicks and from 13.3% to 0.4% on ad-ctr. This is consistent with its role in capturing inter-table dependencies and structural information around the target table. We also observe that the importance of the two modules depends on schema complexity. In HM, which has a simple schema (3 tables and 2 relations), w/o Relation and w/o Fusion yield similar results on user-churn. In contrast, in Avito (8 tables and 11 relations), removing *relation-aware message passing* leads to a much larger drop, where the improvement over the base model is reduced to only 0.4% on ad-ctr. Overall, *relation-aware message passing* is the main driver for modeling complex relational signals, while *context-aware fusion* provides complementary local context.

Encoder Layer and Attention Head. We study the effect of encoder depth and attention head count in the *unified tuple encoder* by varying the number of layers 1,2,3 and heads 1,2,4. Results are shown in Figure 5. The optimal depth depends on schema complexity: a 2-layer encoder works best on Avito (8 tables, 11 relations), while a 1-layer encoder is sufficient on HM (3 tables, 2 relations). It indicates that more complex schemas benefit from higher model capacity, whereas deeper encoders may suffer from overfitting or optimization difficulty in simpler schemas. For attention heads, 2 heads give the best and most stable performance, while more heads bring little benefit but increase unnecessary complexity.

Message-Passing Depth and Aggregator. We next study the depth and aggregator in the *relation-aware message passing* module by varying the depth 1,2,3 and aggregators min, max, sum, mean. Results in Figure 5 show that depth 2 or 3 consistently outperforms depth 1. It is because a larger depth expands the receptive field over the relational graph and captures multi-hop dependencies beyond directly connected neighbors. It reflects that sufficient message-passing depth is necessary to model complex relational structures and obtain expressive tuple representations. Among aggregators, the sum operation performs best overall. Unlike the mean operation, the default choice in most GNNs, it preserves degree-related signals that may be diluted by normalization. Such signals are particularly important in behavior-driven tasks such as user-churn, where the number of neighbors, e.g., user interactions, strongly reflects activity level and helps improve prediction performance.

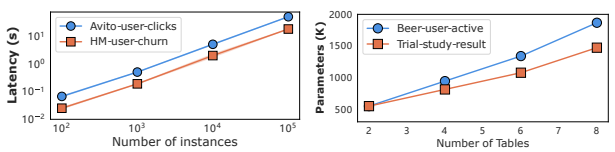
Neighborhood Sampling Configuration. We further study neighborhood sampling in the Dynamic Relation Modeling stage. Since the relational graph can be large, we sample a subset of neighbors $\mathcal{N}(v)$ to efficiently capture structural information. We vary the neighbor size, the hop number, and the sampling strategy, including uniform random sampling and latest, which prioritizes more recent neighbors. Results are shown in Figure 6. Performance generally improves and then stabilizes as the *neighbor size* increases, indicating that larger samples provide richer relational context but diminishing returns. Empirically, a neighbor size of 32 for the user-clicks and 64 for the user-churn is sufficient. The best hop number depends on schema complexity: 2-hop sampling works better on Avito, while 1-hop is more effective on HM. This suggests that more complex schemas benefit from a larger receptive field to capture multi-hop dependencies. Among sampling strategies, latest has the strongest advantage on regression tasks, indicating that temporally recent neighbors often provide more relevant predictive signals.

Relation Importance. Finally, we examine the interpretability of the *context-aware fusion* module by computing recalibrated importance scores for different contextual signals. We rank these signals by their average scores and visualize the top four in Figure 7. The learned importance aligns well with domain intuition. In user-clicks, search receives the highest score, reflecting that user search behavior is a strong indicator of future clicks, while visit is also useful but less influential. In user-churn, user_Info dominates, highlighting



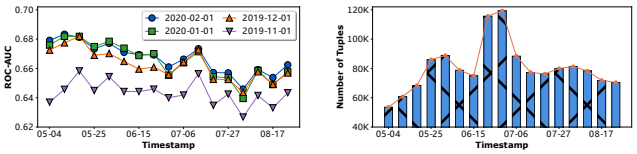
(a) Parameter Size (b) Latency

Figure 8: Evaluation of NeurIDA on computation cost. Numbers on each bar denotes the relative overhead.



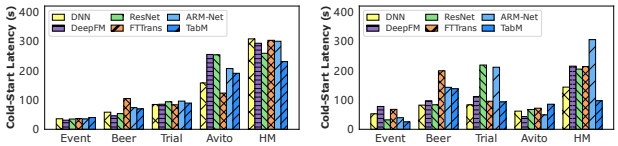
(a) Data Volume (b) Schema Complexity

Figure 9: Evaluation of NeurIDA on scalability with respect to data volume and schema complexity.



(a) Prediction Performance (b) Inserted Tuples

Figure 10: Evaluation of NeurIDA on evolving data for the user-churn task on the HM database.



(a) Classification (b) Regression

Figure 11: Evaluation of NeurIDA on cold-start latency across different base models and databases.

the importance of user profiles, and in item-sales, txn is most important because historical transactions directly relate to future sales. In ad-ctr, search remains dominant, while category (denoted as cat in Figure 7c) also leads to click tendencies. For example, ads in electronics and real estate often attract different levels of user attention and browsing interest. Overall, these results show that NeurIDA not only improves prediction but also provides interpretable insights into which relational signals drive the decision.

5.4 Cost Analysis in Different Scenarios

Parameter Size and Inference Latency. We evaluate the overhead introduced by NeurIDA. First, we choose ResNet as the base model and measure the additional parameters. As shown in Figure 8a, NeurIDA adds between 0.7M and 3M parameters across five databases (excluding initialized embedding tables). The increase follows the complexity of the database schema: more tables and relation types require larger *relation-aware message passing* modules. Overall, the parameter size remains within twice that of the base model. We further measure inference latency, as illustrated in Figure 8b. Across six different base models, NeurIDA introduces a moderate overhead from 1.2 to 1.7 times, compared to the inference latency of base models. Given that NeurIDA eliminates extensive tabular preprocessing stages required by standalone base models, this additional latency remains well within a practical and acceptable range. Overall, NeurIDA delivers significantly better predictive performance with only modest overhead in parameters and latency.

Scalability Analysis. We evaluate the scalability of NeurIDA as data volume and schema complexity increase. For the impact of data volume, we progressively increase the size of the prediction set and measure the inference latency for the user-clicks task in Avito and the user-churn task in HM. As shown in Figure 9a, the inference latency scales near-linearly with the number of prediction samples, indicating efficient and predictable scaling with respect to data volume. To evaluate the impact of schema complexity, we incrementally expand the set of involved tables starting from the target table specified in the *task profile*, and then measure the parameter size of the constructed models for Trial and Beer. Figure 9b shows that the model size increases approximately linearly with the number of tables. The difference in growth slopes is due to their schema complexity, as Beer contains more inter-table relations, therefore requiring additional parameters in the *relation-aware message passing* module. These results demonstrate that NeurIDA can effectively scale to larger prediction workloads and more complex schemas without sacrificing practical efficiency.

Evolving Data and Cold-Start Problem. We study the impact of evolving data and the cold-start latency of NeurIDA. For evolving data, we chronologically partition each database into time windows, treat newly inserted tuples as incremental data, and evaluate prediction performance over time. We also vary the training-data end timestamp to examine the effect of data freshness. As shown in Figure 10a, different end timestamps provide different amounts of historical data for model construction. The results from May to August show that NeurIDA effectively captures general behavioral patterns when sufficient historical data is available, while maintaining stable performance as data evolves.

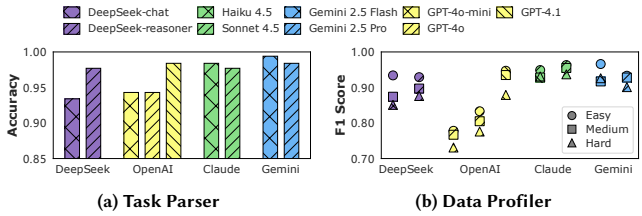
For cold-start problem, we consider a new analytical query for which no task-specific module has been initialized. In this setting, NeurIDA constructs the model by selecting the base model and fine-tuning task-specific modules in DIME. We measure this latency across all databases and base models. As shown in Figure 11, latency varies substantially across databases, mainly due to differences in data volume reported in Table 2, whereas the base model choice has limited impact. Moreover, early stopping terminates fine-tuning once convergence is reached, keeping latency practical. Overall, this analysis confirms that NeurIDA remains effective under evolving data while maintaining acceptable cold-start latency in practice.

5.5 Evaluation of Agent Components

In this subsection, we evaluate nine LLMs from four distinct providers as backends for the agent module in NeurIDA to assess the robustness and compatibility of the system design.

Evaluation of Task Parser. The evaluation set is built from the task profiles in Table 2. Positive samples are generated by varying key attributes and prompting LLMs to produce semantically equivalent natural language queries (NLQs), while negative samples cover representative failure cases, such as ambiguous targets, missing prediction objectives, and schema inconsistencies. The benchmark contains 175 samples, including 120 positive and 55 negative instances, and is evaluated using *accuracy*. As shown in Figure 12a, most frontier models achieve over 94% accuracy, indicating that `Task Parser` is reliable under schema-constrained prompting. Larger models, such as GPT-4.1, generally outperform lightweight models, such as GPT-4o, suggesting that the task benefits from stronger reasoning. Overall, these results confirm that the agent accurately converts natural language analytical requests into structured task profiles for downstream modeling.

Evaluation of Data Profiler. Starting from the five real-world databases in Table 2, we inject controlled schema noise to simulate redundant schemas under three difficulty modes: *Easy*, with a few unrelated noise tables and columns; *Medium*, with partially connected noise tables; and *Hard*, with structurally connected noise tables and additional noise columns. The `Data Profiler` aims to prune these irrelevant injections. We evaluate schema selection using the column-level *F1-score* of the identified task-relevant subset. As shown in Figure 12b, LLM performance degrades as schema complexity increases. In the *Easy* mode, most LLMs achieve F1 scores above 0.93. In the *Medium* mode, performance declines due to partial structural interference, while top-tier LLMs such as Claude remain robust. In the *Hard* mode, the drop becomes more noticeable, but leading models, such as Claude-Sonnet-4.5 and Gemini-2.5, still maintain strong performance. Overall, these results show that the agent-based `Data Profiler` effectively filters irrelevant tables and columns and remains robust under increasing schema complexity.



(a) Task Parser **(b) Data Profiler**

Figure 12: Performance of agent-based components in NeurIDA across nine LLMs from four distinct providers.

6 RELATED WORK

Tabular Data Analytics. Tabular data analytics studies prediction from a single flattened table. Methods such as RF [8] and CatBoost [43] remain strong baselines due to their robustness and efficiency. Recently, deep learning models (e.g., DNN, ARM-Net [10]) have been used to capture nonlinear and high-order feature interactions, especially for high-cardinality categorical features. Meanwhile, tabular foundation models such as TabPFN [22] and TabICL [44] leverage pre-trained priors and in-context adaptation to generalize across tasks. However, these approaches assume single-table inputs and require heavy preprocessing to capture the structure of relational databases. Our work addresses this gap with a unified augmentation framework for existing models.

Table Discovery and Augmentation. Table discovery and augmentation techniques enhance prediction by identifying relevant tables and integrating auxiliary information from data lakes. For example, DLN [7] detects connections across heterogeneous tables via column relevance, while RONIN [37] organizes large data collections hierarchically for efficient exploration. Other work further exploits retrieved tables for augmentation: DFS [23] and AIDA [12] automatically engineer aggregation features from joined data, while Leva [62] uses graph embeddings to preserve relational structure. Unlike data lakes with uncertain relationships, relational databases provide well-defined PK-FK structures. NeurIDA leverages this structure to deliver more accurate and reliable predictions.

In-database ML system. Prior work integrates ML into database systems to reduce data movement and support SQL-based analytics. MADlib [21], LEADS [59], and PostgresML provide in-database ML support. Middleware and declarative frameworks, such as MindsDB [31] and SQLFlow [53], expose external ML engines through SQL interfaces. More integrated systems, like BigQuery ML [17] and Raven [39], support ML execution in database-centered environments. However, these systems are largely limited to static single-table modeling, whereas NeurIDA directly operates on multi-table schemas and dynamically constructs models at query time.

7 CONCLUSIONS

We propose NeurIDA, an end-to-end autonomous system for in-database analytics that dynamically constructs ML models tailored to various analytical tasks. By aligning model design with query intent, relational schema, and task semantics, NeurIDA overcomes the low transferability of static ML models. At its core is dynamic in-database modeling, which constructs models on-the-fly from a shared base model, adapting to diverse prediction objectives without manual pipeline redesign. Additionally, NeurIDA includes LLM-based interfaces for natural language queries and result interpretation. These integrations enable efficient, adaptive, and user-friendly analytics directly within relational databases, paving the way for more intelligent and accessible AI-powered database systems.

ACKNOWLEDGMENTS

We dedicate this work to the memory of Professor Beng Chin Ooi, whose guidance, vision, and generosity have profoundly influenced this research. His legacy will continue to inspire us.

This research is supported by the National Research Foundation, Singapore and Infocomm Media Development Authority under its Trust Tech Funding Initiative, and the National Natural Science Foundation of China (Grant No. 624B2027). Any opinions, findings, or conclusions expressed herein are those of the author(s) and do not necessarily reflect the views of the National Research Foundation, Singapore and Infocomm Media Development Authority.

REFERENCES

[1] 2013. Event Recommendation Engine Challenge. https://www.kaggle.com/c/event-recommendation-engine-challenge. Accessed: 2025-04-16.

[2] 2014. H&M Personalized Fashion Recommendations. https://www.kaggle.com/competitions/h-and-m-personalized-fashion-recommendations. Accessed: 2025-04-16.

[3] 2015. Avito Context Ad Clicks. https://www.kaggle.com/c/avito-context-ad-clicks. Accessed: 2025-04-16.

[4] 2016. AACT Clinical Trials.gove. https://aact.ctti-clinicaltrials.org/. Accessed: 2025-04-16.

[5] Mohamed S. Abdelfattah, Abhinav Mehrotra, Lukasz Dudziak, and Nicholas Donald Lane. 2021. Zero-Cost Proxies for Lightweight NAS. In *9th International Conference on Learning Representations, ICLR*. OpenReview.net.

[6] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In *International Conference on Knowledge Discovery & Data Mining*. ACM, 2623–2631.

[7] Sagar Bharadwaj, Praveen Gupta, Ranjita Bhagwan, and Saikat Guha. 2021. Discovering Related Data At Scale. *Proceedings of the VLDB Endowment* 14, 8 (2021), 1392–1400.

[8] Leo Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (2001), 5–32.

[9] Michael L. Brodie. 1988. Future Intelligent Information Systems: AI and Database Technologies Working Together. In *National Conference on Artificial Intelligence*. AAAI Press / The MIT Press, 844–845.

[10] Shaofeng Cai, Kaiping Zheng, Gang Chen, H. V. Jagadish, Beng Chin Ooi, and Meihui Zhang. 2021. ARM-Net: Adaptive Relation Modeling Network for Structured Data. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 207–220. https://doi.org/10.1145/3448016.3457321

[11] Surajit Chaudhuri. 1998. An overview of query optimization in relational systems. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Seattle, Washington, USA) *(PODS '98)*. Association for Computing Machinery, New York, NY, USA, 34–43. https://doi.org/10.1145/275487.275492

[12] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David R. Karger. 2020. ARDA: Automatic Relational Data Augmentation for Machine Learning. *Proceedings of the VLDB Endowment* 13, 9 (2020), 1373–1387.

[13] David R Cox. 1958. The regression analysis of binary sequences. *Journal of the Royal Statistical Society Series B: Statistical Methodology* 20, 2 (1958), 215–232.

[14] Sanjula De Alwis and Indrajith Ekanayake. 2025. Explainability, risk modeling, and segmentation based customer churn analytics for personalized retention in e-commerce. *arXiv preprint arXiv:2510.11604* (2025).

[15] Matthias Fey, Jinu Sunil, Akihiro Nitta, Rishi Puri, Manan Shah, Blaz Stojanovic, Ramona Bendias, Alexandria Barghi, Vid Kocijan, Zecheng Zhang, Xinwei He, Jan Eric Lenssen, and Jure Leskovec. 2025. PyG 2.0: Scalable Learning on Real World Graphs. *CoRR* abs/2507.16991 (2025).

[16] Rolando Garcia, Pragya Kallanagoudar, Chithra Anand, Sarah E. Chasins, Joseph M. Hellerstein, and Aditya G. Parameswaran. 2024. Flow with FlorDB: Incremental Context Maintenance for the Machine Learning Lifecycle. *CoRR* abs/2408.02498 (2024).

[17] Google Cloud. 2026. BigQuery ML. https://cloud.google.com/bigquery/docs/bqml-introduction. Accessed: 2026-03-23.

[18] Yury Gorishniy, Akim Kotelnikov, and Artem Babenko. 2025. TabM: Advancing tabular deep learning with parameter-efficient ensembling. In *International Conference on Learning Representations*. OpenReview.net.

[19] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. 2021. Revisiting Deep Learning Models for Tabular Data. In *Advances in Neural Information Processing Systems*. 18932–18943.

[20] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *International Joint Conference on Artificial Intelligence*. ijcai.org, 1725–1731.

[21] Joseph M. Hellerstein, Christoper Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, and Arun Kumar. 2012. The MADlib analytics library: or MAD skills, the SQL. *Proceedings of the VLDB Endowment* 5, 12 (2012), 1700–1711.

[22] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. 2023. TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second. In *International Conference on Learning Representations*. OpenReview.net.

[23] James Max Kanter and Kalyan Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *International Conference on Data Science and Advanced Analytics*. IEEE, 1–10.

[24] Konstantinos Karanasos, Matteo Interlandi, Fotis Psallidas, Rathijit Sen, Kwanghyun Park, Ivan Popivanov, Doris Xin, Supun Nakandala, Subru Krishnan, Markus Weimer, Yuan Yu, Raghu Ramakrishnan, and Carlo Curino. 2020. Extending Relational Query Processing with ML Inference. In *Conference on Innovative Data Systems Research*.

[25] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems*. 3146–3154.

[26] Meghdad Kurmanji, Eleni Triantafillou, and Peter Triantafillou. 2024. Machine Unlearning in Learned Databases: An Experimental Analysis. *Proc. ACM Manag. Data* 2, 1 (2024), 49:1–49:26.

[27] Meghdad Kurmanji and Peter Triantafillou. 2023. Detect, Distill and Update: Learned DB Systems Facing Out of Distribution Data. *Proc. ACM Manag. Data* 1, 1, Article 33 (May 2023), 27 pages. https://doi.org/10.1145/3588713

[28] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*. OpenReview.net.

[29] Julian John McAuley and Jure Leskovec. 2013. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *International Conference on World Wide Web*. 897–908.

[30] Ke Meng, Tao He, Sijie Shen, Lei Wang, Wenyuan Yu, and Jingren Zhou. 2025. Moko: Marrying Python with Big Data Systems. In *Proceedings of the European Conference on Computer Systems*. ACM, 345–359.

[31] MindsDB. 2024. https://mindsdb.com/.

[32] Guido Moerkotte and Thomas Neumann. 2008. Dynamic programming strikes back. In *International Conference on Management of Data*. ACM, 539–552.

[33] Supun Nakandala, Yuhao Zhang, and Arun Kumar. 2020. Cerebro: A Data System for Optimized Deep Learning Model Selection. *Proceedings of the VLDB Endowment* 13, 11 (2020), 2159–2173.

[34] Zach Nussbaum, John Xavier Morris, Andriy Mulyar, and Brandon Duderstadt. 2025. Nomic Embed: Training a Reproducible Long Context Text Embedder. *Transactions on Machine Learning Research* 2025 (2025).

[35] Beng Chin Ooi, Shaofeng Cai, Gang Chen, Yanyan Shen, Kian-Lee Tan, Yuncheng Wu, Xiaokui Xiao, Naili Xing, Cong Yue, Lingze Zeng, Meihui Zhang, and Zhanhao Zhao. 2024. NeurDB: an AI-powered autonomous data system. *Sci. China Inf. Sci.* 67, 10 (2024).

[36] Beng Chin Ooi, Kian-Lee Tan, Sheng Wang, Wei Wang, Qingchao Cai, Gang Chen, Jinyang Gao, Zhaojing Luo, Anthony K. H. Tung, Yuan Wang, Zhongle Xie, Meihui Zhang, and Kaiping Zheng. 2015. SINGA: A Distributed Deep Learning Platform. In *Annual ACM Conference on Multimedia Conference*. ACM, 685–688.

[37] Paul Ouellette, Aidan Sciortino, Fatemeh Nargesian, Bahar Ghadiri Bashardoost, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2021. RONIN: Data Lake Exploration. *Proceedings of the VLDB Endowment* 14, 12 (2021), 2863–2866.

[38] James Pan and Guoliang Li. 2025. Database Perspective on LLM Inference Systems. *Proceedings of the VLDB Endowment* 18, 12 (2025), 5504–5507.

[39] Kwanghyun Park, Karla Saur, Dalitso Banda, Rathijit Sen, Matteo Interlandi, and Konstantinos Karanasos. 2022. End-to-end optimization of machine learning prediction queries. In *Proceedings of the 2022 International Conference on Management of Data*. 587–601.

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*. 8024–8035.

[41] PostgresML. 2024. https://postgresml.org/.

[42] Lutz Prechelt. 2012. Early Stopping - But When? In *Neural Networks: Tricks of the Trade - Second Edition*. Lecture Notes in Computer Science, Vol. 7700. Springer, 53–67.

[43] Liudmila Ostroumova Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. CatBoost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*. 6639–6649.

[44] Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. 2025. TabICL: A Tabular Foundation Model for In-Context Learning on Large Data. In *International Conference on Machine Learning*.

[45] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. 1979. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data* (Boston, Massachusetts) *(SIGMOD '79)*. Association for Computing Machinery, New York, NY, USA, 23–34. https://doi.org/10.1145/582095.582099

[46] Yao Shu, Shaofeng Cai, Zhongxiang Dai, Beng Chin Ooi, and Bryan Kian Hsiang Low. 2022. NASI: Label- and Data-agnostic Neural Architecture Search at Initialization. In *The Tenth International Conference on Learning Representations, ICLR*. OpenReview.net.

[47] Yao Shu, Wei Wang, and Shaofeng Cai. 2020. Understanding Architectures Learnt by Cell-based Neural Architecture Search. In *8th International Conference on Learning Representations, ICLR*. OpenReview.net.

[48] Azure SQL. 2024. https://learn.microsoft.com/en-us/azure/azure-sql/.

[49] Michael Stonebraker and Lawrence A. Rowe. 1986. The design of POSTGRES. In *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data* (Washington, D.C., USA) *(SIGMOD '86)*. Association for Computing Machinery, New York, NY, USA, 340–355. https://doi.org/10.1145/16894.16888

[50] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *CoRR* abs/2302.13971 (2023).

[51] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-scale Machine Learning. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 1009–1024. https://doi.org/10.1145/3035918.3064029

[52] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*. 5998–6008.

[53] Yi Wang, Yang Yang, Weiguo Zhu, Yi Wu, Xu Yan, Yongfeng Liu, Yu Wang, Liang Xie, Ziyao Gao, Wenjing Zhu, et al. 2020. Sqlflow: A bridge between SQL and machine learning. *arXiv preprint arXiv:2001.06846* (2020).

[54] Peizhi Wu and Zachary G. Ives. 2024. Modeling Shifting Workloads for Learned Database Systems. *Proc. ACM Manag. Data* 2, 1, Article 38 (March 2024), 27 pages. https://doi.org/10.1145/3639293

[55] Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2024. C-Pack: Packed Resources For General Chinese Embeddings. In *International Conference on Research and Development in Information Retrieval*. ACM, 641–649.

[56] Naili Xing, Shaofeng Cai, Gang Chen, Zhaojing Luo, Beng Chin Ooi, and Jian Pei. 2024. Database Native Model Selection: Harnessing Deep Neural Networks in Database Systems. *Proceedings of the VLDB Endowment* 17, 5 (2024), 1020–1033.

[57] Naili Xing, Shaofeng Cai, Zhaojing Luo, Beng Chin Ooi, and Jian Pei. 2024. Anytime Neural Architecture Search on Tabular Data. *CoRR* abs/2403.10318 (2024).

[58] Jiahuan Yan, Bo Zheng, Hongxia Xu, Yiheng Zhu, Danny Z. Chen, Jimeng Sun, Jian Wu, and Jintai Chen. 2024. Making Pre-trained Language Models Great on Tabular Prediction. In *International Conference on Learning Representations*. OpenReview.net.

[59] Lingze Zeng, Naili Xing, Shaofeng Cai, Gang Chen, Beng Chin Ooi, Jian Pei, and Yuncheng Wu. 2024. Powering In-Database Dynamic Model Slicing for Structured Data Analytics. *Proceedings of the VLDB Endowment* 17, 13 (2024), 4813–4826.

[60] Mark Zhao, Emanuel Adamiak, and Christos Kozyrakis. 2024. cedar: Optimized and Unified Machine Learning Input Data Pipelines. *Proceedings of the VLDB Endowment* 18, 2 (2024), 488–502.

[61] Zhanhao Zhao, Shaofeng Cai, Haotian Gao, Hexiang Pan, Siqi Xiang, Naili Xing, Gang Chen, Beng Chin Ooi, Yanyan Shen, Yuncheng Wu, and Meihui Zhang. 2025. NeurDB: On the Design and Implementation of an AI-powered Autonomous Database. *Conference on Innovative Data Systems Research*.

[62] Zixuan Zhao and Raul Castro Fernandez. 2022. Leva: Boosting Machine Learning Performance with Relational Embedding Data Augmentation. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1504–1517. https://doi.org/10.1145/3514221.3517891

[63] Jiaqi Zhu, Shaofeng Cai, Jie Chen, Fang Deng, Beng Chin Ooi, and Wenqiao Zhang. 2026. Catching every Ripple: Enhanced Anomaly Awareness via Dynamic Concept Adaptation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2026).

[64] Jiaqi Zhu, Shaofeng Cai, Yanyan Shen, Gang Chen, Fang Deng, and Beng Chin Ooi. 2025. In-Context Adaptation to Concept Drift for Learned Database Operations. In *International Conference on Machine Learning*. PMLR, 79699–79726.