東南大學

本科毕业设计（论文）报告

Semantically-Based Data Lake Table Enhancement System

Number: 71121240

Academic

Last Name: Ding Yuming

School School: School of Software Engineering

Major: : Software Engineering

Advisor: Xiong Runqun

Duration: December 2024 — May 2025

2025 Month Day

Chapter 3: System Detailed Design and Implementation

The Semantics-Based Data Lake Table Enhancement System aims to efficiently discover tables within data lakes and intelligently enhance tables queried by users. Through an automated workflow, the system performs multiple functions including table ingestion, semantic modeling, related table discovery, enhancement suggestion generation, and missing value imputation. These five functions collectively improve both the quality of table enhancements and the quality of enhanced tables, supporting downstream analytical tasks.

The system comprises five key processing steps: data preprocessing, storage management, similar table retrieval, table enrichment execution, and missing value imputation.This chapter first analyzes the system's functional requirements, specifically covering five aspects: table preprocessing, data storage, related table discovery, enhancement execution, and missing value filling. It then details the system architecture design, operational environment, and overall processing workflow based on the technical framework outlined in the previous chapter. Finally, it thoroughly discusses the functional design and implementation of the system's constituent modules—the table preprocessing module, related table discovery module, enhancement execution module, and missing value filling module—covering their internal logic and algorithm design.

3.1 Requirements Analysis

3.1.1 Table Preprocessing

Functional Requirement: Table preprocessing achieves efficient representation and similarity modeling of tables within the data lake. During modeling, the system first employs a pre-trained embedding model to vectorize all tables in the data lake, converting each table into a fixed-length vector. It then calculates similarity between tables based on vector similarity.Finally, the system utilizes this similarity to construct an HNSW (Hash-based Nearest Neighbor) graph, enabling efficient retrieval of similar tables and rapid discovery of related tables.

Performance Requirements: The table preprocessing module must demonstrate robust stability and availability, maintaining high performance when handling tables of varying scales. During the vectorization phase, the module must accurately extract semantic features from tables within a reasonable timeframe and encode each table column into a fixed-length vector. Finally, this module must complete similarity calculations and HNSW graph construction within acceptable time and space complexities.

3.1.2 Relevant Table Discovery

Functional Requirements: This module is responsible for finding related tables based on user queries within the system. It must efficiently discover candidate tables with high similarity to the user's query table from the data lake. Based on the query table and parameters provided by the user (such as the number of results k to return), the module should retrieve the *top-k* related tables from the constructed HNSW graph structure, sort them by similarity, and return them for further use by downstream data augmentation modules.

Performance Requirements: Given the large number of tables in the data lake, this module must minimize query time while maintaining high retrieval accuracy to ensure system responsiveness and user experience.Given that the HNSW graph structure incorporates approximate search strategies and inherent randomness during construction and search, this module does not impose strict requirements on query result consistency or persistence. However, it must still deliver robust stability and query performance to balance efficiency and accuracy demands in practical application scenarios.

3.1.3 Enhanced Execution

Functional Requirement: After identifying relevant tables, the enhanced execution module must perform table enrichment operations based on the discovered tables and the user's query table. The module should accurately and reasonably enrich the original table using extracted enrichment parameters and suggestions provided by the LLM, then output the enriched table.

Performance Requirements: This module must ensure high reliability. Given the uncertainty of LLM outputs, it must adhere to two critical requirements: First, strictly process outputs according to predefined formatting specifications; second, reasonably utilize enhancement suggestions provided by the LLM to ensure final results meet requirements. Additionally, it must incorporate robust exception handling mechanisms to correctly address and resolve various issues that may arise during the enhancement process, safeguarding result accuracy and the stable operation of the entire system.

3.1.4 Table Missing Value Filling

Functional Requirements: After enrichment operations, tables may contain missing values. This module must accurately identify the data types of each column and employ appropriate methods to fill in missing values. During filling, it should preserve the original table's semantic and structural characteristics as much as possible, avoiding modifications that alter table semantics and impact the training effectiveness of downstream machine learning tasks. Performance Requirements: The table missing value filling module must meet high processing efficiency and reliability. Functionally, the module must: accurately identify the data types of table columns and apply corresponding filling strategies based on these identifications to reasonably complete missing values. Regarding model invocation, since the module requires calling a pre-trained BERT model when handling text-based missing values, it must ensure the stability of model invocation and the accuracy of results. Furthermore, the module must maintain filling quality while keeping filling speed within an acceptable range.

3.1.5 Data Storage

Functional Requirements: This module manages data across the entire system, requiring unified storage and maintenance of all data, including raw table data in the data lake, vectorized embeddings of tables, and all user-related data. The module must provide a unified external data access interface supporting insert, update, delete, and query operations on all data types. These operations must be encapsulated to ensure other system modules can efficiently and reliably access and manipulate required data.

Performance Requirements: This module must guarantee high reliability in data management, including strict assurance of data integrity, consistency, durability, and fault tolerance. It must prevent data loss or corruption during system anomalies,

and provide continuous, stable data service capabilities.

3.2 Architecture Design

The system adopts a layered architecture design comprising four layers: the interaction interface layer, the business processing layer, the data storage layer, and the physical resource layer. The prototype system design is illustrated in Figure 3-1. The interaction interface layer handles user-system interactions and includes three primary pages: the login/registration page, the table enhancement page, and the downstream task evaluation page. These pages guide users through the table enhancement workflow and effectiveness assessment.The business processing layer forms the system's core, comprising two major modules: relevant table discovery and table augmentation. During relevant table discovery, the system utilizes pre-trained Jina Embedding models and column-wise Transformer models to convert user-uploaded tables into embeddings. It then performs online retrieval of *top-k* tables from the HNSW graph constructed during preprocessing to support augmentation tasks.During the table augmentation phase, the system remotely invokes an LLM to automatically complete table augmentation and missing value filling based on relevant information. The data storage layer is responsible for storing data lake tables and user data. The preprocessing stage stores the converted table vector data, which significantly reduces user online query time and enhances user experience. The physical resource layer includes hardware such as processors and disks, providing fundamental support for system operation.

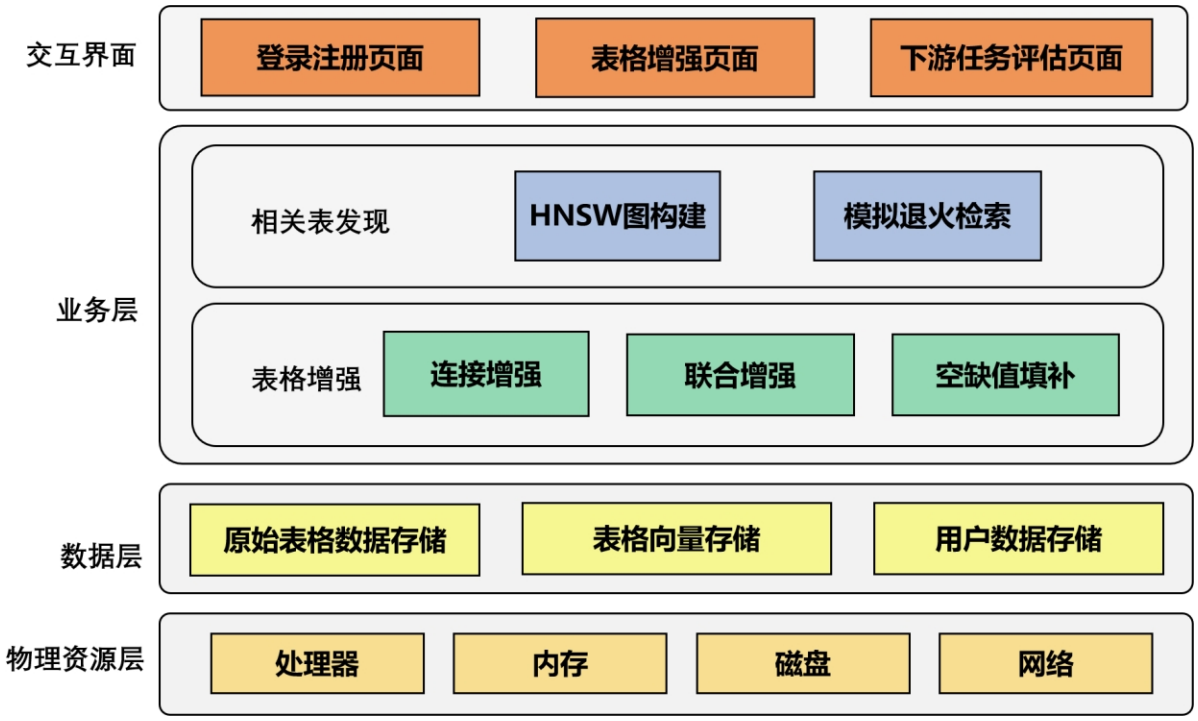


Figure 3-1 System Architecture Design

The system employs a frontend-backend separation architecture in actual deployment. The frontend, built on the Vue3 framework, handles page display and user interaction. The backend, implemented using the FastAPI framework, manages business logic, data storage, and API provisioning. All required pre-trained models are deployed locally on the backend, with code written using mainstream libraries such as PyTorch and Pandas.

Within the backend system, since table enhancement involves four steps—data preprocessing, related table discovery, enhancement execution, and missing value filling—the automation system can be constructed according to each processing step. The backend system architecture is shown in Figure 3-2, divided into two parts: the preprocessing section and the user ad-hoc query section.

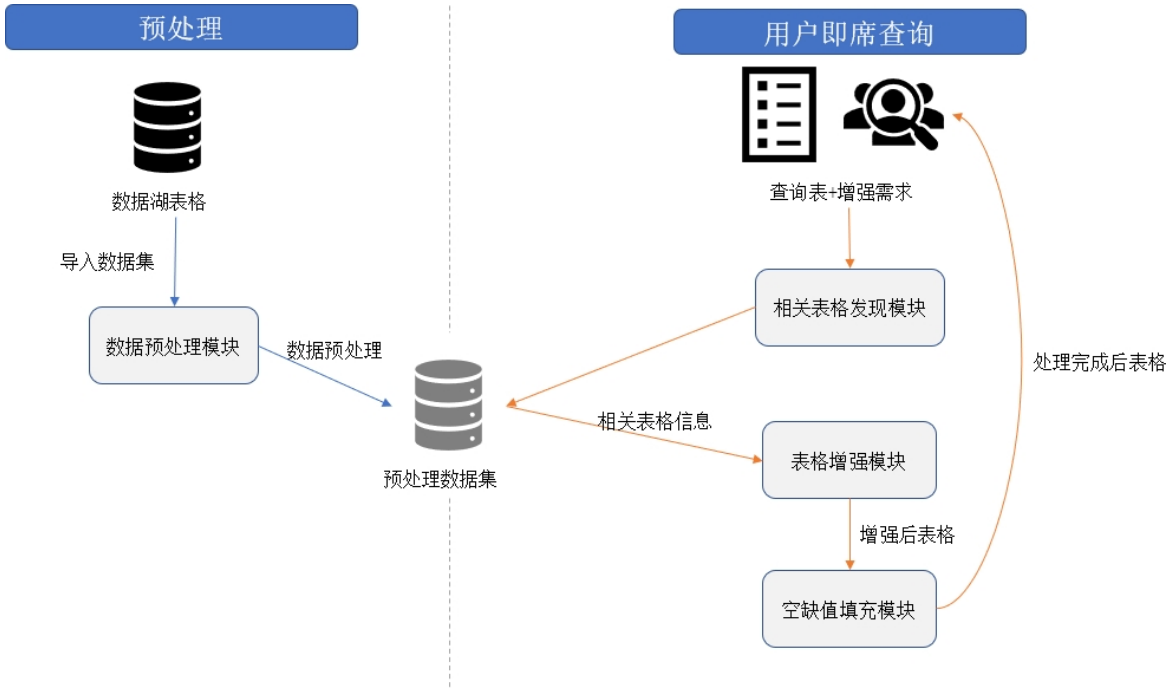


Figure 3-2 Backend System Architecture Diagram

The backend system primarily consists of three functional modules: data preprocessing, relevant table discovery, and table augmentation. To achieve full automation of the table augmentation workflow, the system automatically vectorizes all tables in the data lake during initialization and constructs an index graph based on the HNSW algorithm.

During user ad-hoc queries, the backend system enables efficient module coordination through predefined workflows. First, the relevant table discovery module retrieves table information related to the user query using the constructed HNSW graph and passes it to the table enhancement module.Subsequently, the table enhancement module first performs enhancement operations on the user's table by integrating enhancement requirements, relevant table information, and enhancement suggestions provided by the LLM. It then completes the missing items in the enhanced table. The finalized table is returned to the user as the final result.

3.3 Process Design

3.3.1 Overall Process

The system software flowchart in Figure 3-3 illustrates the backend system architecture design and specific implementation components. The backend system primarily includes the MySQL database, the table vectorization Embedding model, the table enhancement components Pandas and LLM, and the missing value filling BERT model.

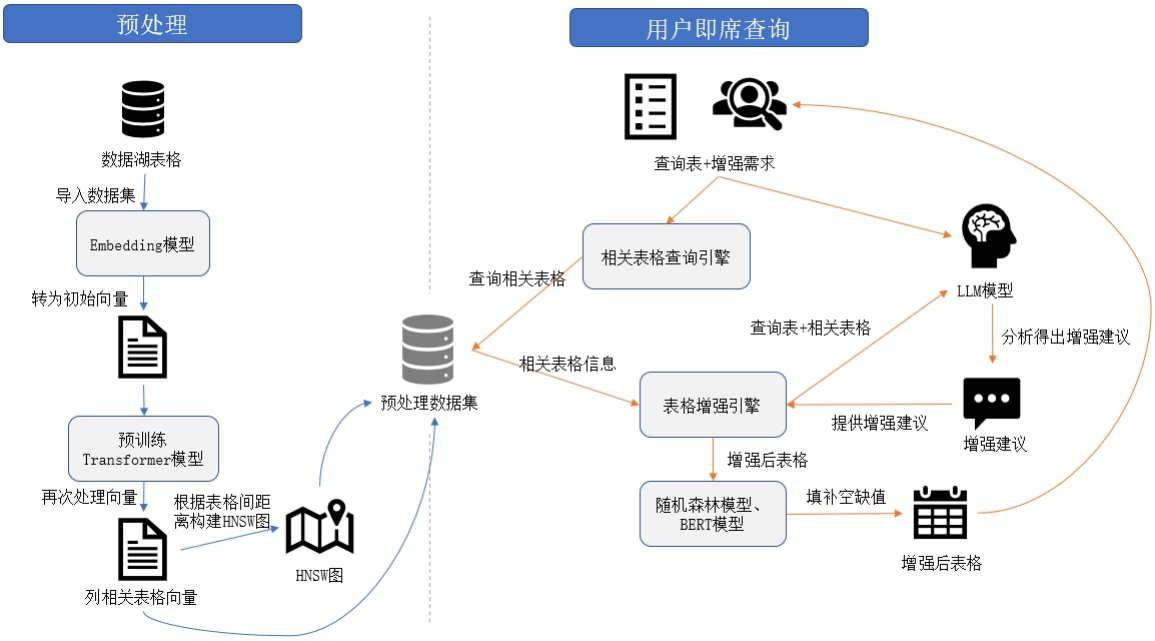


Figure 3-3 Backend System Software Flowchart

3.3.2 Table Preprocessing

3.3.2.1 Overview

The table preprocessing module must run at the initial stage of the entire system operation, converting table data into vector arrays through two transformations. Since text vectorization requires an Embedding model and establishing semantic relationships between table columns necessitates a Transformer model, this module is designed and implemented around these two core models.

Jina Embeddings is a text vectorization model proposed by Jina AI, primarily used to convert text into high-dimensional vectors.The Jina Embeddings model is based on the Transformer architecture and incorporates contrastive learning and multi-task learning techniques during training. This enables strong performance in tasks like semantic text matching and vector retrieval. The model can accept token inputs up to 8192 characters in length while maintaining fast inference speeds and deployability, making it suitable for production-level applications. Error! Reference source not found.

Due to the strong implicit nature of semantic associations between tables and dependencies among columns within tables, modeling these relationships explicitly is challenging. Consequently, traditional supervised learning methods cannot be directly applied when training Transformer models for column-to-column relationships. Therefore, the system employs contrastive learning. By constructing positive and negative sample pairs for training through contrastive learning, the model can autonomously learn semantic relationships between column vectors without explicit labels.

During training, the Normalized Temperature Scale Cross-Entropy Loss (NT-Xent Loss) is employed as the objective function. This loss function enhances the model's ability to distinguish between positive and negative sample pairs while leveraging the Transformer's multi-head self-attention mechanism incorrect

error: no reference found, effectively capturing deep semantic associations between column vectors to improve the accuracy of tabular

semantic representations.

Further enhance the accuracy of the relevant table discovery module.

After obtaining the vector array representing the tables, the system first calculates the overall similarity between different tables using a custom table distance formula. Based on these similarity results, the system constructs an HNSW graph structure to establish similarities between tables. When a user submits a query table, the system can discover relevant tables based on the HNSW graph and employ a simulated annealing-like strategy, providing a high-quality set of relevant tables for subsequent enhancement operations.

3.3.2.2 Table Column Encoding

When converting table columns into vectors, the system initially employed a straightforward approach: concatenating all rows of a column into a continuous text string separated by spaces as input for the embedding model. However, this method has significant drawbacks. Typical pre-trained embedding models often impose input length constraints, such as common models supporting a maximum of 512 tokens. Although the Jina Embedding v2 base-en model supports longer inputs (8192 tokens) error: no reference found, practical applications still encounter issues where certain table column texts exceed this limit, sometimes reaching tens of thousands of tokens. To preserve column semantic features while meeting model input constraints, the system designed a

TF-IDF-based column content truncation method. This approach first traverses all tables to compute the inverse document frequency (IDF) of the global vocabulary. Subsequently, for each row in a table column, it calculates the sum of TF-IDF scores as the row's score. Finally, rows are sorted by descending score and concatenated sequentially until reaching the model's maximum acceptable token length, discarding any remaining content.

This approach effectively controls input length while preserving critical content, ensuring high-quality embeddings. It lays the foundation for subsequent column vector representation and table similarity modeling. Detailed steps of this processing flow are outlined in Algorithm 1.

算法 *1*：获取表格的文本表示

Input: 表格 T, 单词得分字典 IDF, 最长 token 输入长度 m
Output: 表格文本 column_texts

foreach row r ∈ T **do**
  $\text{row\_score}_r = \sum_{\text{token t} \in r} \text{IDF}_t$
column_texts = []
$\text{max\_lenth} = \max(\text{len}(\text{column\_texts}_c))$ **for** column c∈T
while max_lenth < m **do**
  row r = (row has max row_score)
  foreach cell c∈r **do**
    $\text{column\_texts}[\text{name}_c]$.appned(c)
  $\text{row\_score}_r = 0$
  update max_lenth
return column_texts

To validate the effectiveness of the TF-IDF algorithm in preserving tabular semantic features, this paper conducts comparisons across three datasets. Experiments contrast the accuracy of embeddings obtained using TF-IDF-selected rows versus randomly selected rows in relevant tabular discovery tasks. Experimental results are presented in Table 3-1.

Table 3-1 Accuracy Comparison of Table Extraction Algorithms

Dataset	Number of Candidate Tables	TF-IDF	RANDOM
Santos small	1	**1.000**	0.980
	5	**0.812**	0.801
	10	**0.732**	0.724
WWk small	1	0.035	**0.036**
	5	**0.295**	0.245
	10	**0.427**	0.389
WWk large	1	**0.036**	0.035
	5	**0.245**	0.200
	10	**0.386**	0.369

As shown in Table 3-1, the TF-IDF algorithm outperforms the random row selection method across all three datasets, demonstrating its ability to effectively preserve tabular semantic features while reducing input length.

Finally, inputting each column of text into the Embedding model yields a corresponding vector array for each table.

3.3.2.3 Comparative Learning

To more effectively capture semantic features between table columns while accounting for their implicit and associative nature, this paper employs a Transformer model combined with contrastive learning to model table features. The training objective aims to reduce the distance between semantically identical or similar tables while increasing the distance between dissimilar tables.

To achieve contrastive learning, an operation $E(s, n)$ is defined to represent table augmentation. For all tables in the data lake $t \in T$, an augmented copy is generated. After augmentation, each table corresponds to an original table t_{ori} and its augmented copy t_{aug}. Subsequently, following the methodology described in the previous section

, they are respectively converted into vector representations Z_{ori} and Z_{aug}. The steps of the enhancement operation are shown in Algorithm 2, and the comparison before and after table enhancement is shown in Figure 3-4.

before and after enhancement are shown in Figure 3-4.

算法 2：增强单张表格

Input: 表格 T, 交换单元格数量 s, 噪声大小 n
Output: 增强后表格 T

```
for i from 1 to s do
    c = column random chose from T
    c1,c2 = two cells random chose from c
    swap c1,c2
foreach column c∈T do
    if type_c = number do
        inject noise into column c according to noise level n
return T
```

country	capital	population
China	Beijing	1,412,000,000
Canada	Ottawa	40,000,000
Japan	Tokyo	125,000,000

country	capital	population
China	Beijing	1,412,000,123
Japan	Ottawa	40,000,456
Canada	Tokyo	125,000,789

Figure 3-4 Comparison of Tables Before and After Augmentation

Next, the contrastive learning enhancement process trains the Transformer model using a contrastive loss function, as detailed in Algorithm 3.

Algorithm 3.

算法 3：训练列相关 TRANSFORMER 模型

Input: 数据湖表格向量 Z, 增强后数据湖表格向量 Z', 学习率 θ, 每批样本数 b, 训练轮数 e
Output: 训练后模型 model

```
model = Transformer model with random parameter
for ep from 1 to e do
    random split Z,Z' into batches {B_1, ..., B_{|D|/b}}
    for B,B' ∈ {B_1, ..., B_{|Z|/b}} do
        loss = L_contrast(Z_B, Z_B')
        model = backward(model,θ,∂loss/∂model)
return model
```

The loss function is defined as follows. The cross-entropy loss function is employed. For the original table array B and the augmented table B' within the same batch, and for the vectors z_i and z'_i derived from the same table, the single-pair loss $l(z_i, z')$ is defined as

$$l(z_i, z') \triangleq -\log \frac{\exp(simi(z_i, z')/t)}{\sum_{k=1,k\neq i}^{b} \exp(simi(z_i, z_k)/t) + \sum_{k=1}^{b} \exp(simi(z_i, z'_k)/t)} \tag{3.1}$$

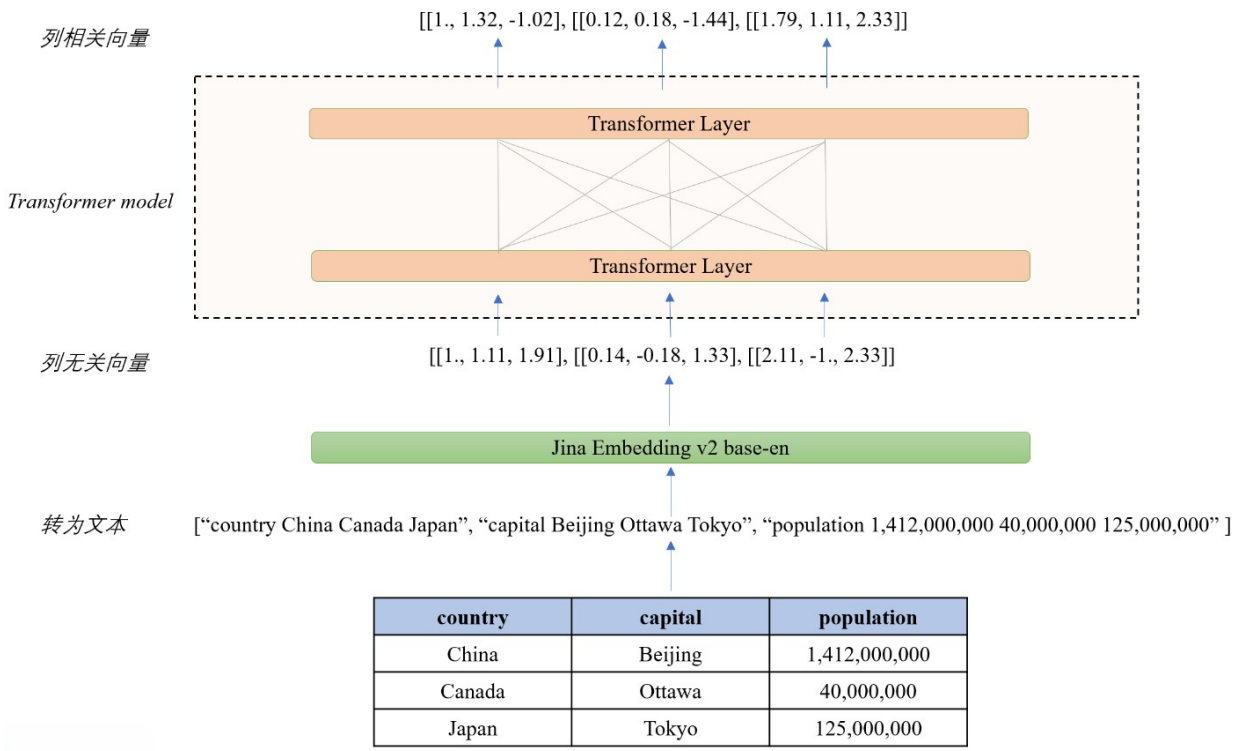
In equation (3.1), *simi* denotes the table similarity function, and t represents the temperature hyperparameter. Minimizing the similarity score between the original vector and the augmented vector within the same table while maximizing the similarity score between vectors across different tables yields *a* smaller *loss*.

For a batch , the overall loss function $L_{contrast}$ is defined as shown in Equation (3.2):

$$L_{contrast} = \frac{1}{2b} \sum_{k=1}^{b} [l(z_k, z'_k) + l(z'_k, z_k)] \tag{3.2}$$

3.3.2.4 Transformer-based Semantic Relevance Models

The proposed Transformer column-semantic correlation model aims to transform column-independent vector arrays into column-correlated vector arrays. During implementation, a specially designed and trained Transformer model with six hidden layers was employed using contrastive learning to perform inter-column semantic modeling tasks. The complete model architecture is illustrated in Figure 3-5.



country	capital	population
China	Beijing	1,412,000,000
Canada	Ottawa	40,000,000
Japan	Tokyo	125,000,000

Figure 3-5 Model Architecture Design Diagram

The input to this Transformer model is a two-dimensional vector array of dimensions$n \times 768$, where n represents the number of columns in the table.After passing through the model's encoder and decoder, the input vector outputs a two-dimensional vector array with the same dimensions:$n \times 768$. By leveraging the Transformer's self-attention mechanism to obtain column-related vectors within a table, the model effectively captures semantic relationships between columns. This generates column semantic correlation vectors, thereby enhancing the accuracy of identifying related tables.

3.3.2.5 Table Similarity

In this module and the related tables module, identifying and determining related tables hinges on calculating similarity between them. To holistically consider semantic relationships across columns and derive an overall similarity score, this paper combines a greedy algorithm to model table similarity calculation as a weighted bipartite matching problem. More specifically, for any two tables

S and T, this paper constructs a bipartite graph$G = \langle S, T, E \rangle$, where S and T represent all columns in the two tables respectively, and the edge set E denotes the similarity scores between any two columns in the tables.To mitigate the interference of dissimilar column pairs on the overall similarity score, for any$s \in S$ and$t \in T$, the edge weight $E(s, t)$ is calculated as shown in Equation (3.3).

$$E(s,t) = \begin{cases} cosine\_similar(T_{i,a}, T_{j,b}) & if\ cosine\_similar(T_{i,a}, T_{j,b}) \geq \beta \\ \text{does not exist} & else \end{cases} \tag{3.3}$$

After obtaining the bipartite graph, to eliminate the advantage of tables with more columns, the similarity score between two tables S and T is calculated as shown in formula (3.4). By dividing by the number of columns in the table with more columns, the advantage of the table with more columns is neutralized.

$$Simi^{(Project)}(S,T) = \frac{max_{macth}G(S,T)}{\max(|S|, |T|)} \quad (3.4)$$

Figure 3-6 illustrates an example of calculating similarity between tables. In this example, the similarity score filter threshold for tables S and T is set to 0.4. Since the column vector similarity between S3 and T3 is less than 0.4, this edge is discarded.

score filtering threshold is set to 0.4. Since the column vector similarity between $S3$ and $T3$ is less than 0.4, this edge is discarded.

In this example, the columns selected for matching are marked with red lines, indicating the similarity scores between the two tables.

For $\frac{0.8 + 0.7 + 0.65}{4}$=0.5375.

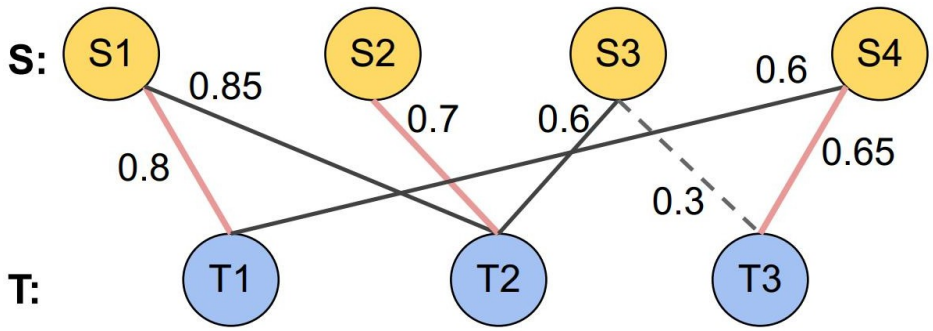


Figure 3-6 Example of Table Similarity Score Calculation

For two tables, directly computing their weighted bipartite graph matching incurs very high time complexity, specifically $O(n^4)$ (converted to network flow computation using Dinic's algorithm) To simplify computation, this paper employs the Sinkhorn-Knopp algorithm, reducing the time complexity for calculating the similarity score of one table pair to $O(n^3)$. The Sinkhorn-Knopp algorithm is an iterative method that transforms a non-negative matrix into a doubly random matrix through alternating row and column normalization. The core idea of the algorithm is to gradually approximate the optimal solution under bipartite random constraints through multiple iterations of row and column normalization operations. Bipartite random matrices are characterized by the sum of each row and column equaling 1, making them significant for bipartite graph maximum matching tasks. By transforming the original similarity matrix into a bipartite random matrix, the weight distribution across rows and columns is effectively balanced. This enables approximate computation of the maximum matching score for bipartite graphs error, noreferencesourcefound. The computational process is detailed in Algorithm 4.

算法 4：获取两张表格的相似度得分

Input: 表格 S, 表格 T, 相似度阈值 β, Sinkhorn-Knopp 迭代次数 num_iter
Output: 表格相似度得分 score

```
for i from 1 to |S| do
    for j from 1 to |T| do
        similarity_matrix[i][j] = cosine_similarity(table1[i], table2[j])
similarity_matrix[similarity_matrix < β] = 0
for k from 1 to num_iter do
    for i from 1 to |S| do
        for j from 1 to |T| do
            similarity_matrix[i][j]/=sum(similarity_matrix[i][*])
            similarity_matrix[i][j]/=sum(similarity_matrix[*][j])
score=0
for k from 1 to min(|S|,|T|) do
    i, j = max(similarity_matrix)
    score+=similarity_matrix[i][j]
    similarity_matrix[i][*]=0
    similarity_matrix[*][j]=0
score/=max(|S|, |T|)
return score
```

3.3.2.6 HNSW Graph Construction

HNSW (Hierarchical Navigable Small World) is a graph structure algorithm designed for approximate nearest neighbor search. Based on a multi-level graph architecture, it achieves efficient search by constructing sparse graphs at multiple levels. The upper-level graph provides coarse-grained navigation, while the lower-level graph handles fine-grained precise search, progressively approaching the optimal solution from coarse to fine granularity.The HNSW graph effectively reduces traversal complexity during search while maintaining high efficiency and accuracy.

Traditional HNSW algorithms primarily address approximate nearest neighbor search between vectors. However, this system employs custom functions for table similarity calculation. To accommodate this specialized similarity metric, this paper modifies the multi-level graph structure.The NSW structure in this paper is based on the minimum spanning tree principle, implemented in three steps: First, data points are probabilistically assigned to different layers. Second, a minimum spanning tree is constructed within each layer, with long and short edges added to enhance connectivity. Finally, nodes with identical identifiers across layers are connected to form cross-layer navigation paths. This design preserves HNSW's advantages of fast and precise search while meeting the requirements of custom similarity calculations. The specific implementation is shown in Algorithm 5.

算法 5：构建 *HNSW-MST* 搜索图

Input: 表格间相似度矩阵 similarity_matrix, 表格数量 n, 图的层数 m, 超参数 κ
Output: HNSW-MST 搜索图 graph

```
graph = []
layer = []
for i from 1 to n do
    for j from 1 to m do
        if rand(0,1)<=κ^(m-j) do
            layer[i] = j
            break
for j from 1 to m do
    current_points = {i∈[1,n] where layer[i]=j}
    g = Minimum Spanning Tree of current_points
    foreach e∈current_points do
        if e∉g and e is a short edge do
            add e to g
        else if e∉g and e is a long edge do
            add e to g
    add g to graph
return graph
```

3.3.3 Related Table Findings

3.3.3.1 Brief Description

The related table discovery module locates relevant tables on the HNSW graph, leveraging inter-table similarity for discovery operations. To enhance efficiency beyond direct traversal of all tables, this paper employs a multi-level graph structure with a nearest neighbor search strategy.The implementation occurs in two steps: First, the system performs layer-by-layer nearest neighbor searches on the HNSW graph, reaching the bottom layer to quickly obtain similarity scores for most potentially related tables, thereby avoiding time-consuming linear traversal. Next, the system ranks all candidate tables by similarity score and outputs the top k highest-scoring tables as the final related tables. These steps ensure result accuracy while significantly improving query efficiency.

3.3.3.2 Algorithm Flow

During a query, the system first randomly selects a starting node from layer 1 and calculates the similarity distance between this node and the query table. At each layer, the system employs a simulated annealing-like strategy: it selects the nearest neighbor node among the current node's neighbors for transition, aiming to approach the local optimum solution for that layer.After completing the search for the current layer, the node with the smallest distance to the query table is selected from all nodes as the starting point for the next layer. This process is executed layer by layer until reaching the bottom layer. Finally, based on the similarity scores of all tables, the top k nodes closest to the query table are selected as the relevant tables to be returned. The pseudocode for the algorithm is shown in Algorithm 6.

算法 6：发现相关表格

Input: 查询表格 q_table, 数据湖表格 tables, HNSW 图 g, 需要返回的相关表格数量 k
Output: 相关表格下标 index

```
distance = [INF]
start = random choose a node from layer 1
distance[start]=Simi(q_table,tables[start])
for i from 1 to g.m do
    steps = |g.layer[i]| / 5
    while steps>0 do
        for v∈neighbour[start] do
            distance[v]=Simi(q_table,tables[start])
        strat = v∈neighbour[start] and distance[v] is minimum
        steps-=1
return index of top-k minimum element in distance
```

3.3.4 Table Enhancement

3.3.4.1 Overview

The table enhancement module comprises two submodules: table enhancement execution and table missing value filling. This module reasonably enhances the user's query table during system operation based on relevant tables and user enhancement requirements.

During the table enhancement execution phase, the system first inputs the user's query table, relevant tables, and extracted enhancement parameters to the LLM to obtain enhancement suggestions.Subsequently, based on the enhancement suggestions output by the LLM, the table enhancement module employs a "JOIN first, then UNION" enhancement logic. It invokes Pandas library functions to enhance the original table and generate the enhanced table. To address the uncertainty of LLM output results, the module incorporates format validation and exception handling mechanisms to ensure the stability of the enhancement process and the correctness of the enhancement results.

During the process of filling in missing values in tables, the system first determines the filling strategy for numeric fields based on enhanced parameters. It then evaluates the data type column by column: if the field is numeric, it applies the filling strategy; if it is text-based, it invokes a pre-trained BERT model to complete the values by considering contextual semantics. This approach enhances data integrity while preserving the original table structure and semantic information as much as possible.

3.3.4.2 Table Augmentation Execution

During table augmentation execution, this system introduces an LLM to achieve more intelligent augmentation decisions compared to traditional methods. The execution phase comprises three main steps. First, the system extracts key information from the user's query table and related tables, including headers and partial data. This information, along with augmentation parameters, is fed into the LLM. Next, the LLM outputs augmentation strategies in a predefined format, containing suggestions for JOIN and UNION operations.Then, the system retrieves and parses the enhancement suggestions provided by the LLM. Finally, based on the parsed results of the enhancement suggestions, the system applies the

predefined enhancement logic to complete semantic-based table structure expansion and table enhancement. The predefined enhancement format recommends two JSON fields: `*join_operations*` and `*union_operations*`.

join_operations is a list where each item is a sublist representing a JOIN operation.The sublist structure can be represented as ["query table column","related table ID.related table column","related table retained column 1","related table retained column 2",...] . For each sublist, the first element is the column from the user query table participating in the JOIN, the second element is the column from the related table participating in the JOIN (including the table ID) and the remaining elements are the columns from the related table that need to be retained in the final enhanced table after the JOIN.`*union_operations*` is a list with an even length. Every pair of adjacent sublists represents a UNION operation. Specifically: - The first sublist specifies the column sources for the UNION, formatted as `related_table_id.column_name`. Multiple columns can be concatenated using `+` to merge them. - The second sublist defines the naming of each column in the final enhanced table, corresponding to the positional order of the columns above.

Here is an example of an enhancement proposal: *{"join_operations": [["department", "1.department", "department", "established date"]], "union_operations": [["2.staff name", "2.department"], ["name", "department"]]}* In the `*join_operations*` , the list `*["department", "1.department", "department", "established date"]*` represents a single table join operation based on the "*department*" field: the system suggests Perform a JOIN operation between the *department* column in the user query table and the *department* column in the related table numbered 1. Retain the *department* and *established date* fields from the related table in the JOIN result. In union_operations, each item list represents a field merge operation. The first item *["2.staff name", "2.department"]* specifies that the staff name and department columns from the related table numbered 2 will be merged into the enhanced table. The second item [*"name"*, "department"] defines the names of these merged columns in the enhanced table, which will be named name and department respectively. 2 as the source columns for the merge. The second entry *["name", "department"]* specifies their names in the final enhanced table as *name* and *department*, respectively.

Upon receiving enhancement suggestions, the system executes the enhancement operations. It first processes the JOIN enhancement, parsing each suggestion to extract query columns, related table IDs, JOIN columns, and columns to retain. The system uses Pandas' merge function to join the related tables with the query table while cleaning redundant columns.Next, the system processes the UNION operation, parsing each pair of operations as a group and uniformly setting new column names. It extracts data from multiple related tables and concatenates them, finally merging the concatenated result into the original query table to complete the data enhancement. Pseudocode for the JOIN and UNION operations is shown in Algorithm 7 and Algorithm 8.

算法 7：执行 *JOIN* 增强操作

```
Input: 查询表格 query_table, JOIN 操作 join_operations, 相关表格 related_tables
Output: 增强后表格 query_table
foreach join_op in join_operations do
    query_column = join_op[0]
    related_ref = join_op[1]
    (table_id, related_column) = ParseTableRef(related_ref)
    keep_columns = Union(join_op[2:], [related_column])
    related_table = LoadCSV(related_tables[table_id])
    related_table = FilterColumns(related_table, keep_columns)
    related_table = DropDuplicates(related_table, related_column)
    query_table = MergeTables(query_table, related_table, on_left = query_column, on_right =
    related_column, method = "outer")
    query_table = DropDuplacateColumns(query_table)
return query_table
```

算法 8：执行 *UNION* 增强操作

```
Input: 查询表格 query_table, UNION 操作 union_operations, 相关表格 related_tables
Output: 增强后表格 query_table
for i from 1 to |union_operations| step=2 do
    operation_list = union_operations[i]
    column_names = union_operations[i + 1]
    involved_tables = ExtractTableIDs(operation_list)
    table_dfs = InitializeEmptyDictionary()
    foreach table_id∈involved_tables do
        table_dfs[table_id] = LoadCSV(related_tables[table_id])
    for j from 1 to |operation_list| do
        op = operation_list[j]
        new_col = column_names[j]
        if '+'∈op do
            ref1, ref2 = SplitJoinColumns(op)
            ApplyJoinBetween(table_dfs, ref1, ref2, new_col)
        else do
            tid, col = ParseTableRef(op)
            RenameColumn(table_dfs[tid], col, new_col)
    union_table = JoinUnionTables(table_dfs, operation_list)
    query_table = ConcatenateRows(query_table, union_table)
return query_table
```

3.3.4.3 Table Null Value Filling

After table augmentation, issues such as inconsistent field alignment and data gaps often result in empty cells within the augmented tables. To enhance data integrity and usability, this system designs and implements a missing value filling module to address missing values in the augmented tables.The system automatically identifies and fills empty fields based on user-specified filling strategies, ensuring the enhanced tables exhibit superior performance in subsequent downstream machine learning tasks.

tasks.

The missing value filling process consists of two main steps. First, the system attempts to convert each column to a numerical type for unified subsequent processing.Next, for non-numeric columns, the system replaces missing values with [MASK] and performs filling using a pre-trained BERT model. For numeric columns, filling is executed according to the strategy specified in the augmentation parameters, which include prediction using a random forest regression model, or filling with the mode or median. Pseudocode for the missing value filling process is shown in Algorithm 9.

算法 9：空缺值填补

```
Input: 待填补表格 df, 填充策略 strategy
Output: 填补后表格 df
foreach column ∈ df do
    convert column to numeric type if possible
foreach column ∈ df do
    if column is not numeric do
        foreach value∈column and value=NA do
            filled_value = BERT_Predict(value)
            value = filled_value
foreach column ∈ df do
    if column is numeric do
        if strategy == "MODEL" do
            use RandomForestRegressor to predict missing values
        else if strategy == "MEAN" do
            fill missing values with mode
        else do
            fill missing values with median
return df
```

During the missing value filling process, the system fills text columns first, followed by numerical columns. This design is based on the fact that text columns often influence the meaning of numerical columns. In structured tabular data, the values of numerical fields frequently depend on text fields. Therefore, filling text columns first provides more complete contextual information for numerical column filling, thereby improving the accuracy of numerical predictions.

3.3.5 Data Storage

3.3.5.1 Overview

The Data Storage module serves as the core component for data storage and management within the entire system. It handles two primary data types: information from data lake tables and all user-related data. This section provides a detailed, categorized explanation of data storage methods across subsystems, integrating insights from previously discussed modules.

The system's storage methods are primarily divided into two categories: structured data managed using MySQL databases, and unstructured or large file data stored via the file system. The former is mainly used to store individual

small individual data points with high overall volume and stringent real-time requirements for CRUD operations, such as user information, data lake table information, and enhanced historical records of user tables.This data typically demands high concurrent access and is efficiently managed through relational databases. The latter is suitable for storing data types with larger individual sizes and lower operational speed requirements, such as vector information in data lake tables and user profile images. The file system better accommodates the capacity and access pattern demands of such data.

3.3.5.2 MySQL Data Storage

Within this system, the MySQL database contains three tables: the user information table, the data table index information table, and the user enhanced history information table. A database representation example is shown in Figure 3-7.

#	username	password	avatar_path	user_type
1	122828[illegible]3@qq.com	64d09d9930c8ecf79e5...	122828[illegible]3@qq.com_...	user
2	admin	5a38afb1a18d408e6cd...	(NULL)	admin

(a) **User Information Table**

#	table_id	table_path	pure_embedding_path	processed_embedding_path
1	1	E:/DLES_System/da...	E:/DLES_System/da...	E:/DLES_System/datalake_e..
2	2	E:/DLES_System/da...	E:/DLES_System/da...	E:/DLES_System/datalake_e..
3	3	E:/DLES_System/da...	E:/DLES_System/da...	E:/DLES_System/datalake_e..
4	4	E:/DLES_System/da...	E:/DLES_System/da...	E:/DLES_System/datalake_e..
5	5	E:/DLES_System/da...	E:/DLES_System/da...	E:/DLES_System/datalake_e..
6	6	E:/DLES_System/da...	E:/DLES_System/da...	E:/DLES_System/datalake_e..
7	7	E:/DLES_System/da...	E:/DLES_System/da...	E:/DLES_System/datalake_e..
8	8	E:/DLES_System/da...	E:/DLES_System/da...	E:/DLES_System/datalake_e..

(b) **Data Table Index Information Table**

#	username	history_tree
1	122828[illegible]@qq.com	{"id": 0, "faid": -1, "label": "1228280263@qq.com", "isFile": false, "chil..

(c) **User Enhancement History Information** Table Figure 3-7 Database Diagram Example

The User Data Table primarily stores basic user information, including account names, passwords, and avatar URLs It is primarily accessed in the system's login and registration modules and handles access control and user authentication in other modules. Data lake table index information is utilized in the data preprocessing module, related table discovery module, and table enhancement module.

Usage. During the data preprocessing phase, the system must write the auto-incrementing ID of the table, the storage path of the data lake table, and the table vector path into the data lake table information table. In the subsequent table discovery and table enhancement phases, querying this table enables rapid retrieval of the data lake table path or table vector path, thereby achieving efficient data location and access based on the file system.

The User Enhancement History Information Table contains user IDs and users' enhancement history information.

Historical information fields adopt

JSON-type fields to simulate hierarchical storage structures within the file system. When users modify their history, this field synchronizes updates with corresponding files in the file system.

3.3.5.3 File System Data Storage

The file system stores data lake tables, table vectors, and user table enhancement records. Data lake tables and their vector files reside in uniformly named folders, while user enhancement records are organized by username under directories structured as "fixed-name folder/username."

Data lake tables must be prepared before system operation. Vector generation involves two steps: First, initial table vectors are generated using the Jina Embedding model and temporarily stored in a directory. Then, pre-trained column-specific Transformer models convert column-agnostic vectors into column-specific vectors, writing them to folders for subsequent module use.

During HNSW index graph construction in the data preprocessing module, the system retrieves vector paths from the database and loads corresponding vector data from the file system to compute table similarity and build the index graph. In the table augmentation module, the system reads relevant data lake tables based on table IDs provided by associated table modules to execute final augmentation operations.

User enhancement history files are also stored in the file system. Each use case is represented as an independent folder containing four files: the raw table used by the current use case, its corresponding initial vector, column-related vectors, and the dialogue history between the user and the LLM. A typical directory structure for an enhancement use case is shown in Figure 3-8.

Figure 3-8 Example of a Table Augmentation Record

3.3.6 Downstream Task Validation

3.3.6.1 Overview

To facilitate user comparison of pre- and post-augmentation results, the system incorporates a downstream task validation module. This module integrates a random forest model to assess the quality of user-uploaded tabular data. Users simply upload data and specify the columns for prediction. The system automatically partitions the data into training and test sets, completes model training and prediction, and outputs the prediction results.

"y prediction. For numerical prediction columns, the root mean square error is quantified using the formula:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

predicted values and actual values. For text-type prediction columns, the F1 score is calculated using the formula: F1 Score = $2 \times \frac{Precision \times Recall}{Precision+Recall}$ to quantify the deviation between predicted and actual values. Upon completion of the evaluation, the system will output the results.

in a bar chart format to help users understand the performance changes brought by table augmentation. Since this module is not the core focus of this paper, we will not elaborate further here.

3.4 Chapter Summary

This chapter systematically introduces a semantic data lake table augmentation system, covering system requirements analysis, system architecture design, and backend architecture. For the five key modules, it provides detailed analysis from functional to performance dimensions, clarifying specific requirements for each.

Building upon this foundation and integrating the technical framework proposed in Chapter 2, this chapter designed four primary system modules: data preprocessing, related table discovery, table enhancement, and missing value filling. It automates the table enhancement process using HNSW graphs and enhancement suggestions provided by LLMs.Furthermore, this chapter specifies the technical framework and runtime environment relied upon by the system, including Python+Vue3, along with the rationale for selecting key technological components such as Jina Embedding and the Qwen large language model. Simultaneously, it details the implementation logic of core processes—including table vectorization, semantic enhancement, and missing value imputation—through software flowcharts.

Building upon the technical framework introduction, this chapter details the implementation logic of the system's three core modules. The table preprocessing module utilizes pre-trained embedding models and fine-tuned Transformer models to convert raw tables into high-quality column-related vector arrays. This module constructs an HNSW graph based on table similarity to support efficient approximate nearest neighbor retrieval.The Related Table Discovery module employs simulated annealing search strategies across layers of the HNSW graph. This strategy iteratively selects and jumps to the nearest node within the current node's neighborhood, approximating local optima layer by layer to ultimately return the *top-k* related tables. The Table Augmentation module comprises two phases. The first phase is Augmentation Execution, where LLM-generated enhancement suggestions are applied to enrich tables following a ''JOIN first, UNION later'' sequence.The second stage involves missing value filling. This step automatically selects appropriate filling methods for numeric and text fields based on user requirements and field types, completing the entire table enhancement process.

Chapter 4: System Testing and Results Analysis

This chapter conducts comprehensive functional and performance testing on the Semantically-Driven Table Augmentation System for Data Lakes (SDASDL), building upon the preceding system design and implementation. This testing validates the effectiveness of the proposed method and the feasibility of the theoretical findings.

4.1 Test Objectives and Configuration

4.1.1 Test Objectives

This section systematically evaluates the Semantically-Driven Table Augmentation System for Data Lakes (SDASDL) to validate its performance in functional completeness, query efficiency, validity of augmented results, and overall augmentation speed. Testing primarily focused on the following five aspects:

(1) Functional Testing

Covering core functional modules such as login/registration, related table discovery, table augmentation, and downstream task evaluation, automated scripts simulate user operations to validate system functionality correctness and consistency.

(2) Related Table Discovery Performance Testing

On three distinct datasets, the proposed related table direction method is compared against baseline methods such as BM25 and TF-IDF. Key evaluation metrics include average search time, precision, and recall.

(3) Table Enhancement Performance Testing

This test focuses on evaluating the effectiveness of downstream task enhancement. Experiments were conducted using the WWk small dataset. Performance comparisons in regression prediction tasks validated the superior effectiveness of the LLM-based intelligent enhancement method over traditional approaches.

(4) Response Efficiency Testing

This test evaluates the system's response speed in ad-hoc interaction scenarios. It measures the end-to-end duration from the user issuing an enhancement command to the completion of table enhancement. Multiple tests were conducted to calculate the average duration, with the default number of relevant tables set to k=10.

(5) Stability Testing

This test examines the overall performance trends of downstream tasks when varying the number of *top-k* tables used in the table augmentation module. It evaluates the system's ability to analyze changes in input-related tables and identifies underlying causes.

4.1.2 Test Configuration

The system's test configuration comprises multiple critical components. Table vectorization employs the Jina Embedding v2 base-en model

, while data storage and processing rely on MySQL and Pandas. To enhance tabular data quality and expressiveness, the system integrates Pandas, Qwen 2.5-max, and DeepSeek-V3 as table augmentation components. For handling missing values,

The system employs scikit-learn alongside Google BERT (bert-base-uncased) as the preprocessing tools. The contrastive learning module utilizes a base model comprising six Transformer layers and eight attention heads as its backbone architecture.During training, the batch size was set to 32, the learning rate to 5e-5, and the maximum token sequence length to 512. All experiments were conducted on a single computer equipped with an NVIDIA RTX 3050 GPU and an Intel i5-12500H CPU. Due to limited local computational resources, the LLM required for system operation was accessed via remote API calls rather than deployed locally.

During testing, the FastAPI service was first launched in the backend environment using uvicorn. Subsequently, the Vue application was run in development mode via the npm run dev command in the frontend environment, completing the overall deployment and startup of the system.

Three datasets were employed to evaluate the system.The first is the Santos small dataset, constructed from the Canadian Open Data Platform. Ten tables were selected as raw data, processed to generate approximately 1,530 structured tables. The second and third are the WWk series datasets, built from tables within Wikipedia pages. First, highly structured tables were filtered from Wikipedia pages, requiring at least 10 rows
data points and 4 valid columns to ensure sufficient information content. Subsequently, for each query table, at least 5 highly relevant target tables were identified and annotated as related tables, ensuring each query table had sufficient related tables throughout the experiment.

To accommodate experiments of varying scales, the WWk dataset is divided into two subsets: WWk small contains approximately 550 tables, while WWk large contains approximately 3,500 tables. This enables simultaneous evaluation of performance and accuracy in both small-scale and large-scale scenarios.

4.2 System Testing and Analysis

4.2.1 Functional Testing

Automated scripts simulate user operations, covering all functional modules to confirm correct system operation. The system correctly performs core tasks including login/registration, related table discovery, table augmentation, and downstream task evaluation. Test results confirm all SDASDL modules function normally with expected outputs, supporting large-scale data lake table augmentation tasks.

4.2.1.1 Login and Registration Module

As shown in Figure 4-1, to enable user identification and authentication, the SDASDL system requires users to complete login or registration before use. For forgotten passwords, the system provides an email-based password recovery method to ensure system availability and enhance user experience. Scripts simulated user registration, login, and password recovery operations, verifying that the system correctly processes user requests and returns accurate results. Users can successfully log in after registration or password recovery, validating the correctness and usability of the login/registration module.

correctly process user requests and return accurate results. After registration or password recovery, users can successfully log in to the system, confirming the correctness and availability of the login and registration module.

(a) Login System

(b) Register Account

(c) Password Recovery

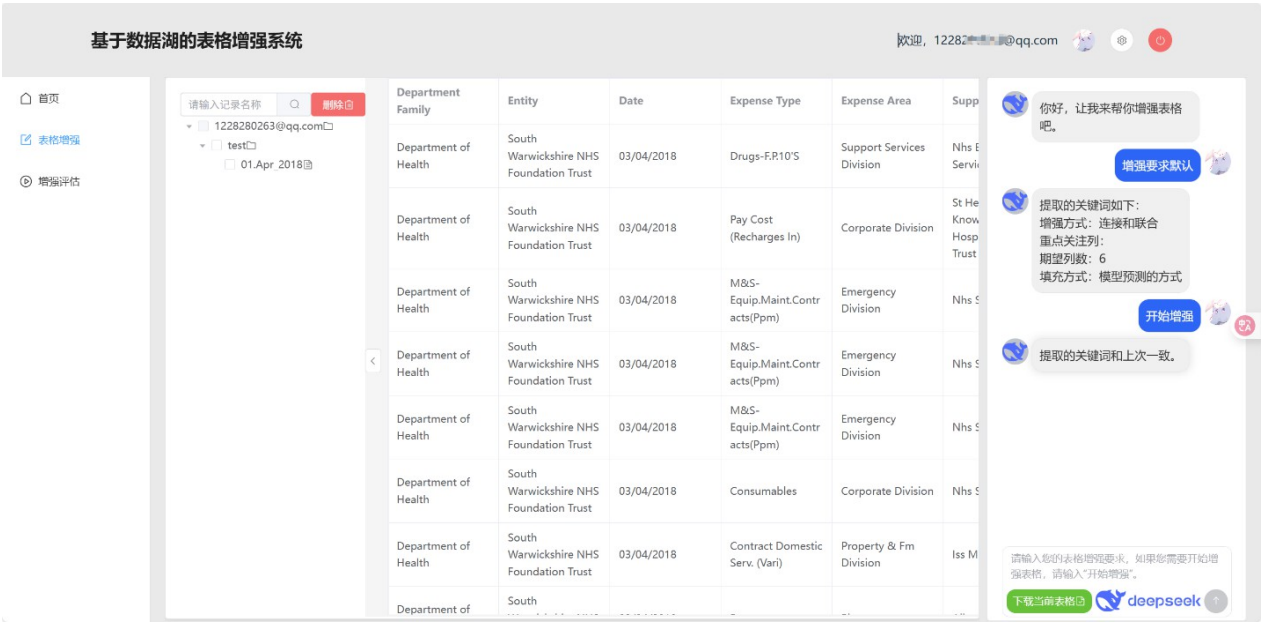
Figure 4-1 System Login Module Interface

4.2.1.2 Form Enhancement Module

The Table Enhancement Module assists users in performing table enhancement operations. The system offers two methods for querying tables: uploading new tables or selecting historical enhancement records. As shown in Figure 4-2, the interface is divided into three main areas: the left pane displays historical records, the center pane shows the specific content of the selected query table, and the right pane provides a dialogue window for interacting with the LLM.

Figure 4-2 Table Enhancement Module Interface

Users interact with the model through a dialogue window, where the model automatically analyzes requirements and generates



enhanced keywords. After confirming the keywords

, the user inputs the "Start Enhancement" command to execute subsequent enhancement operations. The system first identifies relevant tables and then performs the enhancement. Upon obtaining the enhanced tables, the system immediately updates the interface, displaying the latest table results in the central area.This process automates everything from user requirement extraction to result display. Based on the table enhancement flowchart (Figure 4-3), a script was developed to test the table enhancement module. This validated the module's stability when processing queries of varying sizes and its ability to correctly handle exceptions during the enhancement process.

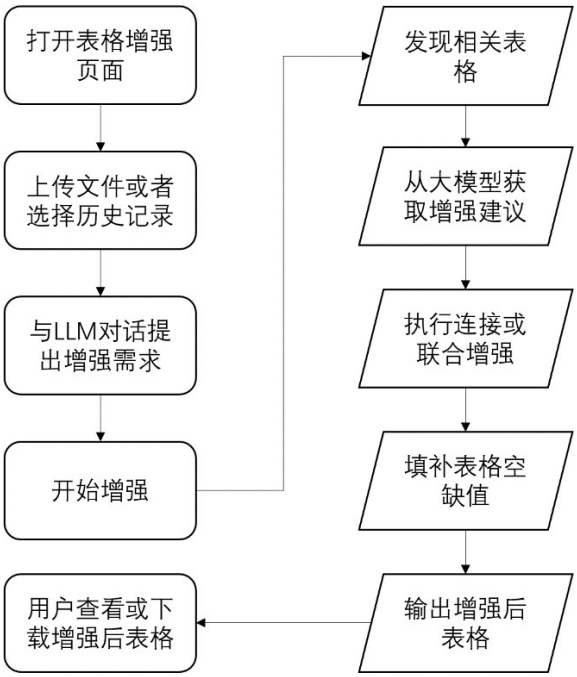


Figure 4-3 Table Enhancement Flowchart

4.2.1.3 Downstream Task Evaluation Page

The Downstream Task Evaluation module incorporates two core functions: First, the system embeds an XGBoost model within this module for predictive task evaluation, enabling users to perform quantitative analysis of enhancement effects. Second, the module integrates LLM code generation capabilities. If users find the system's default model parameters insufficient for evaluation needs, they can interact with the LLM to obtain customized code, achieving more flexible evaluation configurations and experimental comparisons.

The Multi-Table Prediction Evaluation page is shown in Figure 4-4. This page allows users to upload up to 5 tables in the "Table Upload" area. All uploaded tables appear in the "Uploaded Files" list, where users can also delete tables.In the "Training Parameters" section, users can input various model training parameters and select a common column across all tables as the target prediction column. Upon clicking the "Start Evaluation" button, the system automatically selects evaluation metrics based on the prediction column's data type. For numerical data, the system uses Root Mean Square Error (RMSE) as the evaluation metric; for textual data, it employs F1-Score.

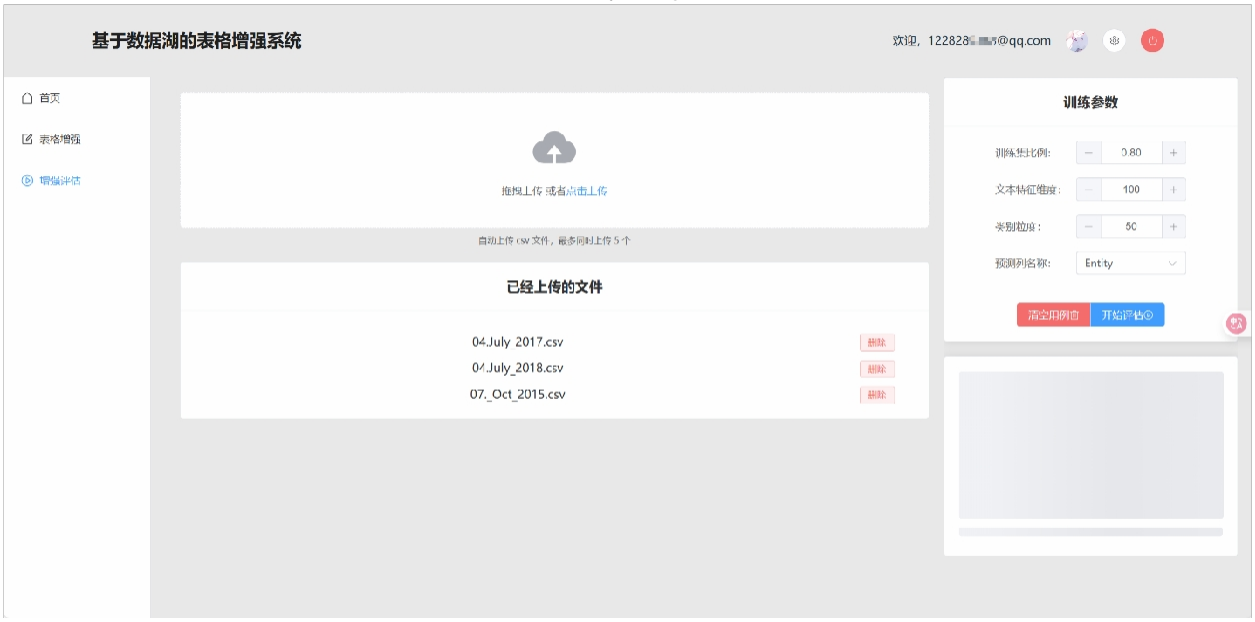


Figure 4-4 Multi-Table Prediction Evaluation Page

Upon completion of the evaluation, the system generates bar charts displaying the evaluation results for each table, as shown in Figure 4-5. This visualization facilitates comparative analysis for users.

Figure 4-5 Evaluation Results Page

Based on the downstream task evaluation flowchart (Figure 4-6), automated scripts were developed to simulate user operations such as adding/deleting tables and adjusting evaluation parameters. This validated that the downstream evaluation module can correctly perform model training and result prediction for tables of varying types and sizes, and accurately return prediction results to users. This successfully verified the module's correctness and its ability to handle exceptional cases appropriately.

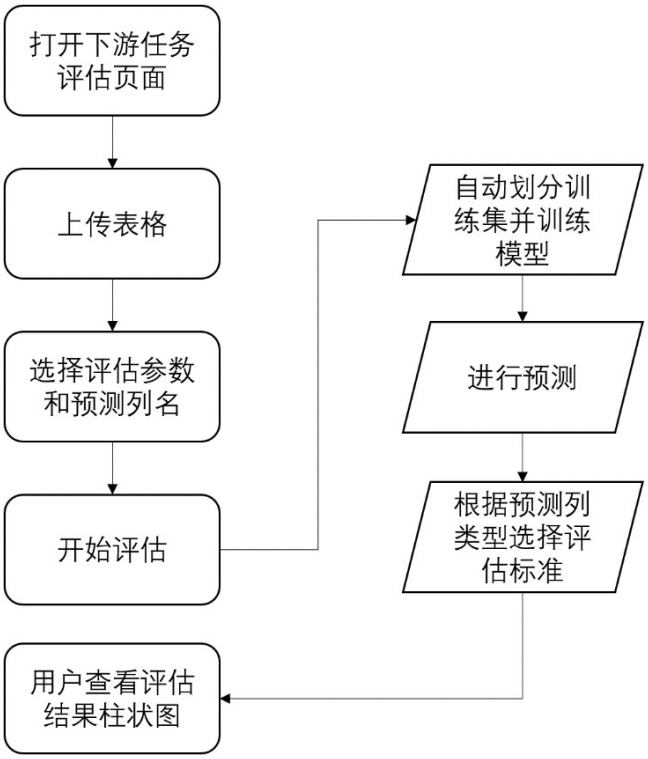
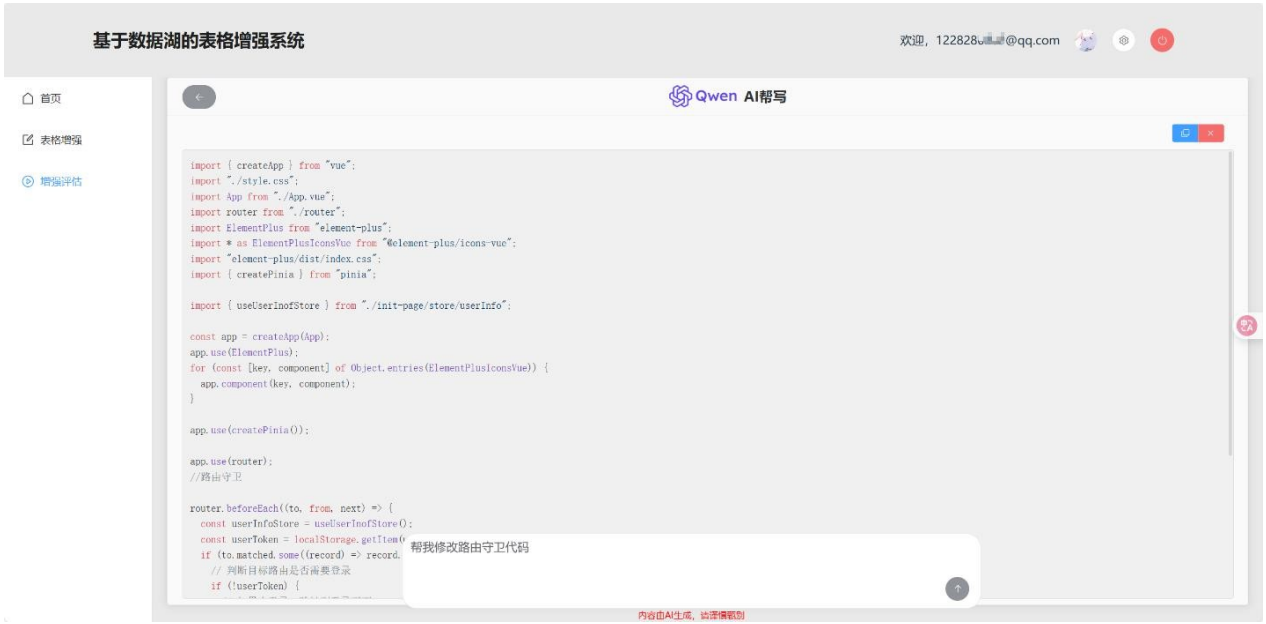


Figure 4-6 Downstream Task Evaluation Flowchart

For additional user requirements, the system offers an "AI Code Assistance" feature to address personalized needs, as shown in Figure 4-7. Users can input existing code into the code input box and specify detailed coding requirements in the request input area. Upon clicking the "Upload" button, the system invokes the LLM to generate code based on user needs and input code, finally displaying the results in the code input box to help users rapidly complete code writing.

Figure 4-7 AI Assisted Writing Page



4.2.2 Related Tables Performance Testing

4.2.2.1 Benchmark Comparison

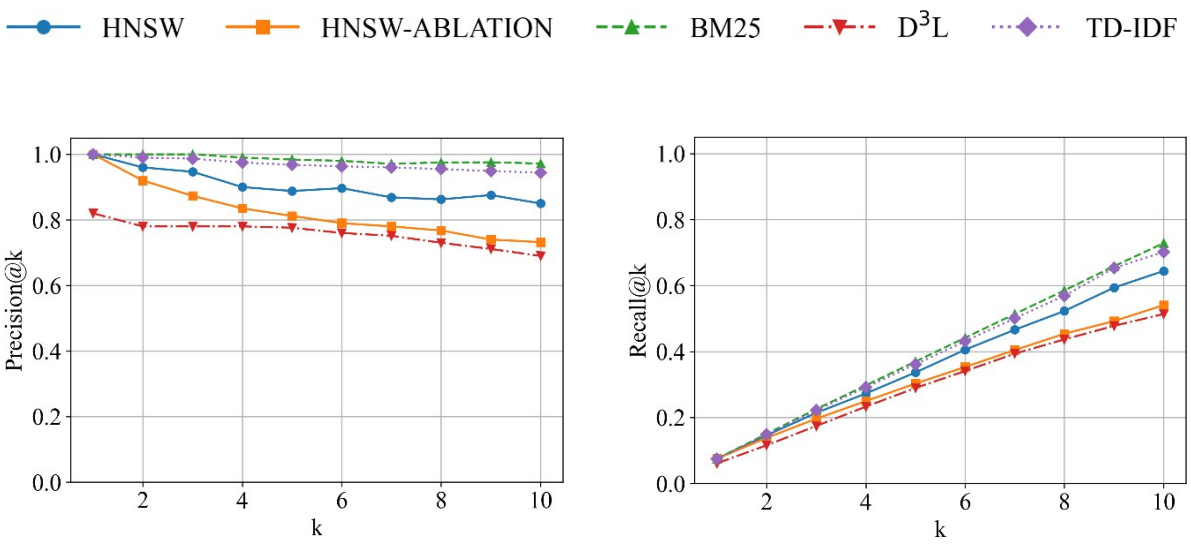
In the related table discovery performance test, the HNSW-MST method proposed in this paper was compared with the following benchmarks.

- TF-IDF: A feature representation method based on inverse document frequency weighted by term frequency within local tables and global document occurrence, used to measure table similarity.
- BM25: A scoring function based on a probabilistic retrieval model that optimizes table similarity accuracy by incorporating term frequency saturation effects and column length normalization.
- D³L: A table federated search method that identifies related tables through multiple dimensions, including column name overlap, value overlap, and format similarity.

Experiments compare the proposed method with various baseline approaches. Key evaluation metrics include precision and recall, alongside average search speed across different datasets. These comparisons validate the proposed method's advantages in both accuracy and efficiency.

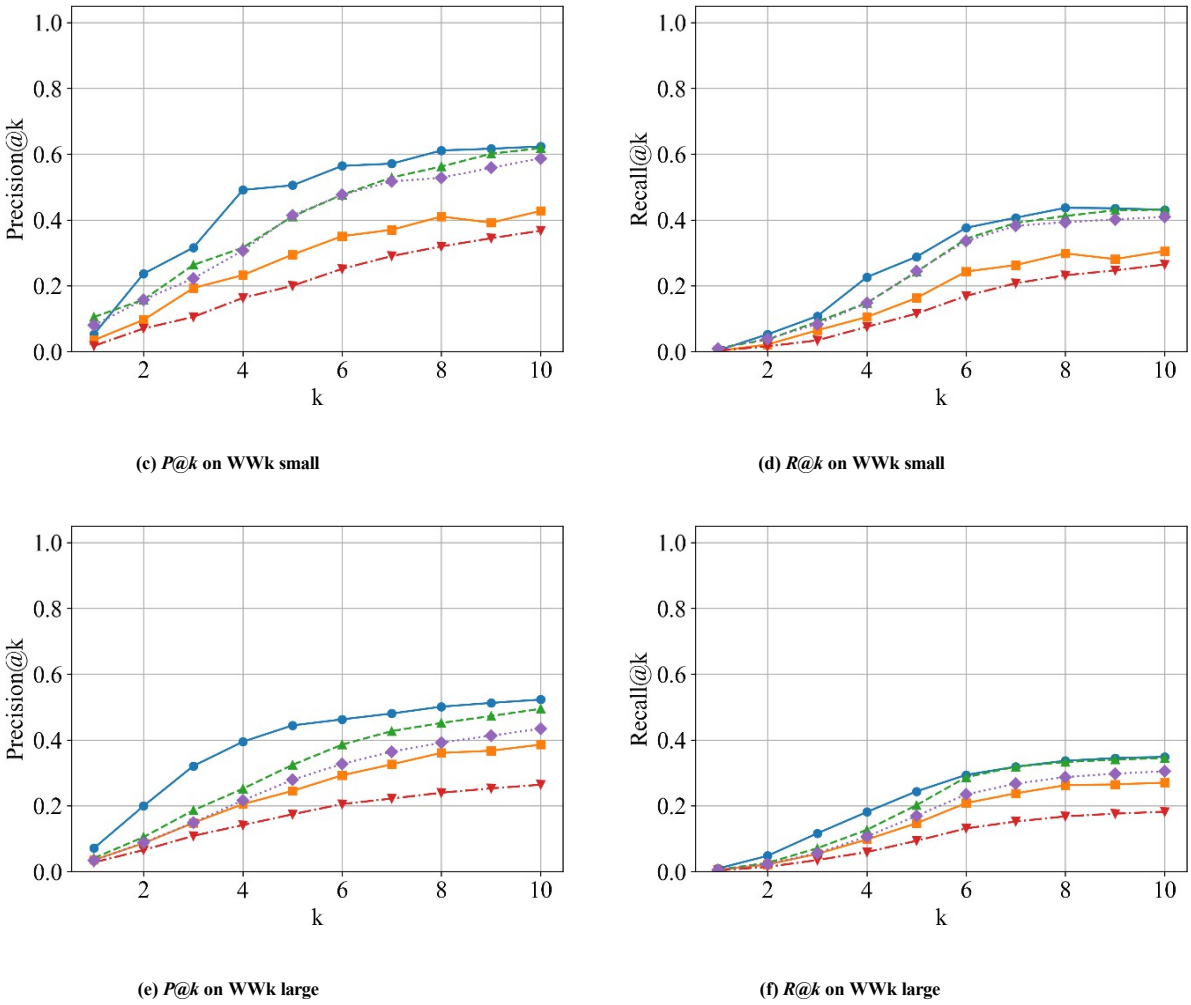
4.2.2.2 Test Results

Figures 4-8 illustrate the comparison of precision and recall between our method and baseline methods for the related table discovery task across three distinct datasets.



(a) *P@k* on Santos small

(b) *R@k* on Santos small



(c) *P@k* on WWk small

(d) *R@k* on WWk small

(e) *P@k* on WWk large

(f) *R@k* on WWk large

Figure 4-8 Comparison of Accuracy and Recall in Relevant Table Discovery

Figure 4-8 shows that on the WWk dataset, the proposed HNSW-based related table discovery method achieves approximately 10% higher accuracy and recall compared to BM25, the best-performing baseline method. On the Santos dataset, however, our method performs slightly worse than both BM25 and TF-IDF.This discrepancy likely stems from the simpler construction process of the Santos dataset. The Santos dataset was created by randomly splitting large tables into smaller ones, resulting in a high proportion of duplicate fields among related tables. Consequently, text-matching-based related table discovery methods like BM25 and TF-IDF exhibit certain advantages on this dataset.Conversely, in the WWk dataset, tables originate from Wikipedia pages and are primarily linked through semantic rather than textual matching. Since the proposed method more effectively captures semantic relationships within and between tables, it demonstrates superior performance over BM25 and TF-IDF on this dataset.

Furthermore, ablation experiments validate the importance of column-level semantic modeling. Comparisons between the full HNSW method and the HNSW-ABLATION method (which removes the column-level semantic modeling Transformer model) reveal that the full model outperforms the ablation version on all three datasets. Specifically, average gains in accuracy and recall exceed 20%. This difference conclusively demonstrates that column-level semantic modeling significantly enhances the accuracy of related table discovery.

all three datasets. Specifically, average improvements in precision and recall exceeded 20%. This disparity conclusively demonstrates that column-level semantic modeling substantially enhances the accuracy of related table discovery.

Regarding query latency, the proposed related table retrieval method significantly outperformed baseline approaches across all datasets. It achieved an average query latency reduction exceeding 50% compared to other methods, demonstrating exceptional speed advantages. Detailed query latency results for each dataset are presented in Table 4-1.

Table 4-1 Average Query Latency (s)

Method Name	Santos small	WWk small	WWk large
HNSW	**0.0213**	**0.0157**	**0.1492**
D^3L	0.7800	0.2959	0.6137
BM25	0.3055	0.0360	0.3301
TF-IDF	0.3515	0.0757	0.5782

Table 4-1 demonstrates that the proposed method outperforms all baseline methods in query speed. On the Santos dataset, the proposed method is 90% faster than the fastest baseline; on the WWk dataset, query latency is reduced by over 50%.The above data demonstrates the method's significant speed advantage across datasets of varying scales. Analysis of table sizes across datasets indicates that the Santos dataset features larger average tables, leading to higher query latency for text-based methods like BM25 and TF-IDF. Conversely, the smaller tables in the WWk dataset enable relatively faster query speeds for BM25 and TF-IDF methods.

4.2.3 Table Enhancement Performance Testing

4.2.3.1 Comparison Benchmarks

In the table enhancement performance test, the proposed LLM-based intelligent enhancement method is primarily compared with the following table enhancement approaches.

(1) Simple Baseline Join: This most fundamental join strategy involves directly joining tables based on randomly selected columns with identical names.

(2) Statistical Baseline Join: Selects columns with the highest similarity for joining based on column-level similarity.

(3) Random Filtered Join: After excluding columns with excessively high or low unique values, randomly selects join keys for a left join.

(4) Column Name Matching Baseline Join: Determines join columns by calculating string similarity between column names.

In the table augmentation comparison experiments, the proposed LLM-based intelligent augmentation method was evaluated against various baseline methods on regression prediction tasks. Specific metrics included Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Coefficient of Determination (R^2). Comparative analysis of these error metrics demonstrates the effectiveness and superiority of the proposed method in enhancing the performance of downstream prediction tasks.

(RMSE), Mean Absolute Error (MAE), and Coefficient of Determination (R^2). Comparative analysis of these error metrics demonstrates the effectiveness and superiority of the proposed method in enhancing downstream prediction task performance.

4.2.3.2 Test Results

Table 4-2 presents the average improvement in evaluation metrics across downstream regression prediction tasks for augmented tables generated using different table augmentation methods on the WWk small dataset.

Table 4-2 Improvement in Prediction Metrics (%)

Method	MSE	RMSE	MAE	R^2
LLM-Based Intelligent Enhancement	**46.15**	**46.15**	**46.15**	**65.38**
Simple Baseline Connection	19.23	19.23	23.07	15.38
Statistical Baseline Connection	3.84	3.84	3.84	3.84
Random Filter Connection	19.23	19.23	19.23	23.07
Listed Matching Baseline Connection	7.69	7.69	7.69	7.69

Experimental results demonstrate that the LLM-based intelligent enhancement method outperforms other baseline methods in regression prediction tasks. As shown in Table 4-2, the proposed method achieves significant improvements across four key metrics: a 46.15% reduction in MSE, RMSE, and MAE, and a 65.38% increase in R^2, substantially outperforming the second-best simple baseline connection method.Traditional approaches like statistical baseline joining and column name matching baseline show limited improvement, indicating that joining strategies relying solely on column names or unique value ratios are ineffective. In contrast, the proposed method significantly enhances prediction performance by leveraging LLM to deeply understand semantic relationships between tables, fully demonstrating its advantages in table augmentation.

4.2.4 Response Performance Testing

To evaluate the response performance of the SDASDL system in practical scenarios, this section examines how the overall system and individual module processing times vary with increasing numbers of tables requiring augmentation. Results are shown in Figures 4-9. The experimental setup employs the proposed HNSW-like method to accelerate table discovery within the relevant table discovery module, utilizes top-10 relevant tables for each augmentation in the table augmentation module, and scales the number of test tables incrementally from 50 to 500.

Test results indicate that the total processing time of the SDASDL system exhibits a linear growth trend with the number of tables to be enhanced, increasing from 15 seconds to 155 seconds. The average processing time per table is approximately 0.31 seconds, and the total processing time curve largely overlaps with the table enhancement time curve.This indicates that the system's efficiency bottleneck primarily occurs during the enhancement phase, while the time required for discovering related tables is negligible at the millisecond level. In interactive real-world scenarios, enhancing a single table takes less than 10 seconds, meeting system usability requirements. Experimental results demonstrate that the SDASDL system, in its current implementation, already achieves conventional end-to-end latency

and usability.

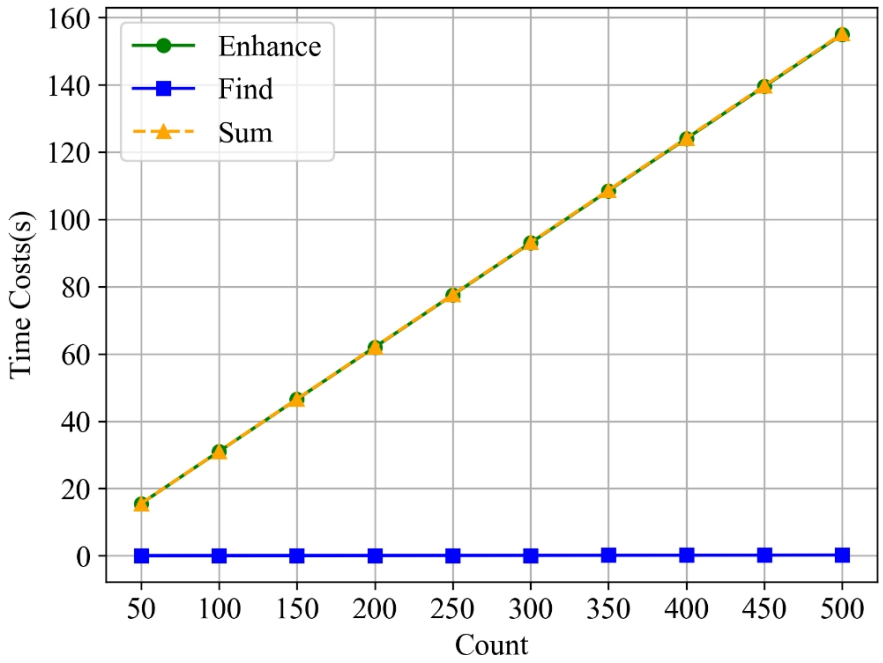
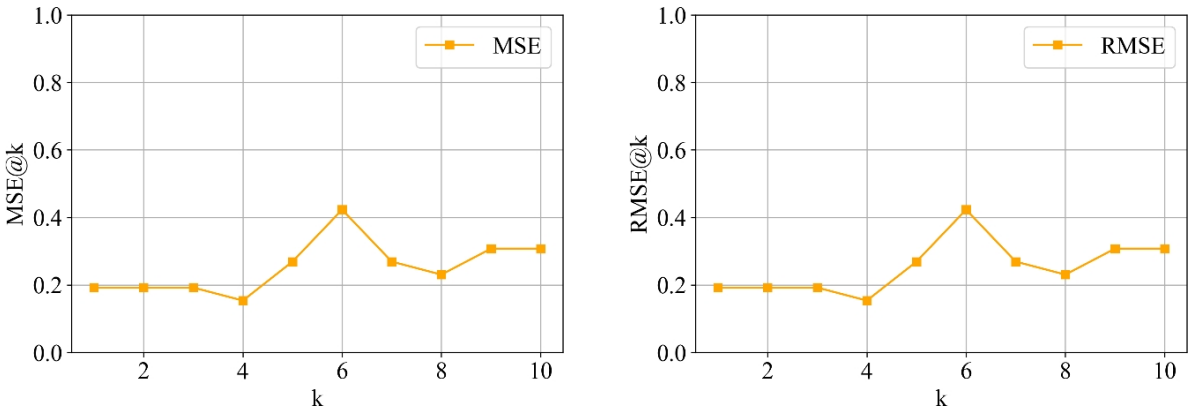


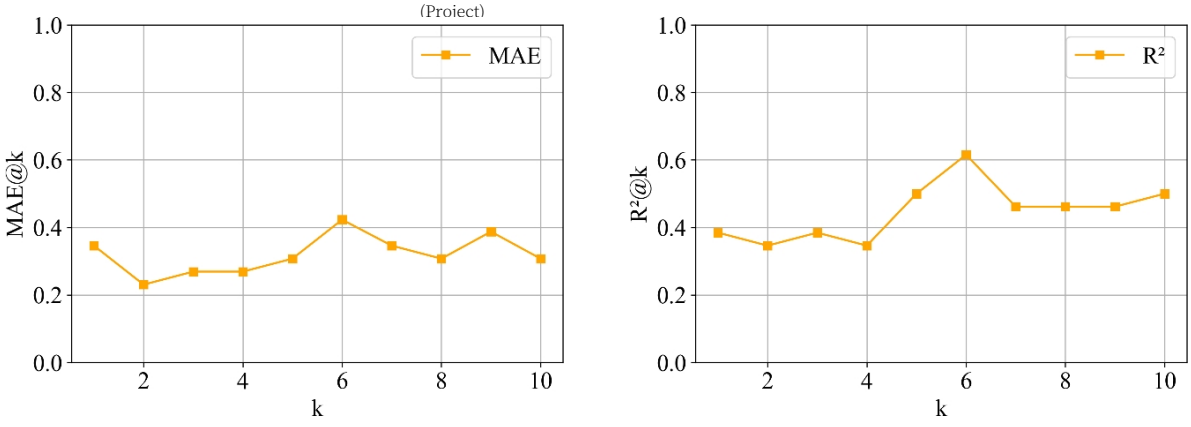
Figure 4-9: Time consumption curve of the SDASDL system under different table quantities

4.2.5 Stability Testing

To evaluate the impact of the number of selected relevant tables *k* during the table enhancement phase on enhancement effectiveness, this study tested the variation in four performance metrics of downstream machine learning tasks as the number of candidate tables increased from 1 to 10. The results are shown in Figure 4-10.



(a) MSE **(b) RMSE**



(c) MAE (d) R²

Figure 4-10 Performance of table augmentation using different numbers of related tables during the table augmentation phase

Overall, the enhancement performance curve exhibits a trend of " " initial slight fluctuations followed by an upward rise and then a stable decline,"with peak enhancement performance occurring at k=6. Detailed analysis is as follows: When $k \leq 4$, all four metrics are relatively low with minor fluctuations, indicating insufficient relevant table information available at this stage and poor table enhancement effectiveness;When k increases to 6, the overall system performance achieves its optimal effect. At this point, more information becomes available, and the LLM provides the best enhancement suggestions under this information volume. When k continues to increase to 10, the introduction of excessive redundant information overloads the LLM's input. The overly long input impairs the LLM's decision-making capability, leading to a decline in enhancement performance at this stage.

Comprehensive analysis reveals that the enhancement performance of the SDASDL system exhibits a distinct inflection point characteristic with respect to the k value: a small number of relevant tables ($k \leq 4$) provide limited information, offering only basic improvements; as k continues to increase, the effect improves, reaching a peak at k=6; when $k \geq 7$, excessive redundant information impairs the LLM's decision-making, leading to a decline in enhancement performance.Therefore, k=6 is set for practical use in this system.

4.3 Chapter Summary

This chapter evaluates the proposed related table discovery and table enhancement method through multidimensional experiments. Results demonstrate significant advantages in semantically complex scenarios: on the WWk dataset, accuracy and recall improve by approximately 10% compared to the optimal baseline BM25, while query latency decreases by over 60%.Ablation experiments demonstrate that the column-level Transformer model improves relevant table discovery accuracy by over 20%, proving the importance of inter-column semantic modeling. In downstream regression task evaluations, the LLM-based intelligent enhancement strategy performs excellently, achieving over 40% improvement across all four metrics and significantly outperforming traditional column name matching methods. The above findings fully validate the comprehensive advantages of the proposed method in terms of accuracy, efficiency, and applicability.

Concurrently, this chapter systematically introduces the design and implementation of the Semantically Driven Data Lake Table Augmentation System (SDASDL),

demonstrating automated and intelligent table augmentation through efficient query mechanisms and judicious LLM utilization. The system employs a layered architecture with frontend-backend separation to ensure scalability and operational efficiency. It comprises login/registration, relevant table discovery, table augmentation, and downstream evaluation modules. Subsequently, this chapter evaluates system performance through multiple tests, verifying SDASDL's robust functional integrity, usability, and accuracy throughout the end-to-end table augmentation workflow. This demonstrates the effectiveness and practicality of the proposed method. Experimental results confirm that SDASDL enhances downstream machine learning task performance while maintaining system stability, validating the feasibility of the proposed approach in real-world data lake environments.

心求至善

SOUTHEAST UNIVERSITY