2025 IEEE Twelfth International Conference on Intelligent Computing and Information Systems (ICICIS)

DataForge: An AI-Driven Data Warehouse Schema Generator

Abdelrahman A. Abdelrahman
Information Systems Department, Faculty of Computer and Information Sciences
Ain Shams University
Cairo, Egypt
2021170298@cis.asu.edu.eg

Abdelrahman A. Elbahrawy
Information Systems Department, Faculty of Computer and Information Sciences
Ain Shams University
Cairo, Egypt
2021170296@cis.asu.edu.eg

Ahmed R. Sobieh
Information Systems Department, Faculty of Computer and Information Sciences
Ain Shams University
Cairo, Egypt
2021170023@cis.asu.edu.eg

Alaa E. ElSaid
Information Systems Department, Faculty of Computer and Information Sciences
Ain Shams University
Cairo, Egypt
2021170079@cis.asu.edu.eg

Ahmed M. Ali
Information Systems Department, Faculty of Computer and Information Sciences
Ain Shams University
Cairo, Egypt
2021170049@cis.asu.edu.eg

Arwa A. Elsharawy
Information Systems Department, Faculty of Computer and Information Sciences
Ain Shams University
Cairo, Egypt
2021170058@cis.asu.edu.eg

Yasmine Shaaban
Information Systems Department, Faculty of Computer and Information Sciences
Ain Shams University
Cairo, Egypt
yasmine.shaban@cis.asu.edu.eg

Yasmine M. Afify
Information Systems Department, Faculty of Computer and Information Sciences
Ain Shams University
Cairo, Egypt
yasmine.afify@cis.asu.edu.eg

Abstract— **Organizations today rely on robust data warehouses to integrate and analyze massive volumes of data for decision-making. However, designing an optimized warehouse schema (fact and dimension tables, keys, constraints, naming conventions) is a labor-intensive, error-prone process that can take weeks of manual effort by experts. This paper presents DataForge, an AI-driven framework that automates and accelerates data warehouse schema creation. DataForge parses input SQL schema definitions, infers an appropriate star schema, and enhances it using AI. It combines regex-based and grammar-based SQL parsing to reliably extract tables, columns, and relationships; keyword-driven domain detection and NLP techniques to infer business context; semantic validation to enforce logical consistency and naming standards; and heuristic rules to classify tables into fact or dimension roles. An interactive web-based interface allows users to visualize and refine the generated schema with real-time suggestions. In benchmark evaluations on retail, healthcare, and financial datasets, DataForge reduced schema design time by over 80% and achieved an expert-validated schema quality score above 90%. The results indicate that DataForge can dramatically streamline the schema design phase, paving the way for faster, more consistent data engineering workflows.**

Keywords—schema automation, dimensional modeling, NLP, BERT, Tokenization.

I. Introduction

In the era of big data, enterprises across industries depend on data-driven decision making for competitive advantage. Data warehouses (DWs) serve as centralized repositories that aggregate data from heterogeneous sources (transactional databases, logs, APIs) and support complex analytical queries and business intelligence (BI) reporting. Designing a high-quality DW schema—organizing data into fact tables and dimension tables with proper keys and relationships—is critical to enable efficient Online Analytical Processing (OLAP). Traditionally, this schema design process is performed manually by data engineers following methodologies like dimensional modeling (star/snowflake schemas) [1]. Manual design involves carefully parsing source database schemas, identifying entities and relationships, defining primary/foreign keys, and enforcing naming conventions and consistency.

This process is *time-consuming* and requires deep domain expertise: large organizations report significant delays in analytics projects due to the slow turnaround of manual data integration and schema design [2]. It is also *error-prone*—mistakes such as missing foreign key constraints or inconsistent naming can lead to inaccurate analyses or costly rework. Moreover, as data sources and business requirements evolve, maintaining the schema becomes increasingly difficult, often necessitating multiple revisions or even parallel data marts for different use cases. These challenges motivate the need for automation in DW schema design.

Recent advances in artificial intelligence and automation offer an opportunity to fundamentally improve this process. In particular, natural language processing (NLP) and pattern recognition can be applied to understand schema metadata, while machine learning heuristics or even large language models can capture domain knowledge and best practices. Some early research efforts have explored aspects of this problem. For example, Usman et al. developed one of the first frameworks for automatically generating OLAP schemas from data by applying data mining techniques [3].

DiScala and Abadi proposed algorithms to infer normalized relational schemas from nested data sources without human intervention [4]. Others have introduced rule-based approaches to derive dimensional models by analyzing database usage patterns and foreign key relationships. More recently, researchers have applied advanced AI methods: Mali et al. present a cost-based optimization framework (FACT-DM) to automatically transform data models under resource constraints [5], and Dwivedi et al. demonstrate that graph neural networks can learn relational structure, generating schemas by treating databases as graphs. NLP-driven techniques have also been employed; for instance, Imarisio et al. fine-tuned a BERT language model to capture semantic similarities in database metadata for data discovery tasks [6], and Salem leveraged large language models (LLMs) to interpret natural language requirements into database schemas.

Commercial data integration tools (e.g., Informatica, Talend) provide limited automation in schema design, focusing mainly on Extract, Transform, Load (ETL) workflows or template-based modeling, and lack AI-driven insights. In summary, prior works either tackle specific sub-problems or require extensive training data, and few offer an end-to-end automated solution that incorporates user feedback.

DataForge is designed to fill this gap by providing a holistic, AI-augmented platform for automated data warehouse schema generation. The key insight behind DataForge is that a combination of lightweight parsing techniques, heuristic rules, and learned intelligence can drastically reduce the manual effort while preserving expert-level accuracy.

DataForge accepts an input of one or more raw SQL DDL (Data Definition Language) files describing source tables. It then proceeds to: (1) parse the SQL definitions to extract all tables, columns, data types, and constraints; (2) classify and organize these into an initial DW schema (identifying which tables are candidate fact tables versus dimension tables, inferring primary keys and foreign key relationships between them); and (3) enhance the schema using AI-driven analysis to ensure it aligns with the business domain and follows best practices (adding any missing surrogate keys, slowly changing dimension support, timestamp fields, etc.). All these suggestions are presented to the user via an interactive interface where further manual adjustments can be made with real-time validation [7].

The contributions of this work are fourfold as follows:

- An automated schema parsing and generation pipeline that combines regex-based and grammar-based SQL parsing with heuristics for fact/dimension classification. This pipeline can rapidly produce a star schema draft from raw DDL with minimal human input.
- Integration of domain detection and NLP to bring semantic awareness into schema design. DataForge uses a keyword-based technique augmented by a BERT-based model to infer the business domain (e.g., retail, healthcare) from table and column names, enabling domain-specific recommendations (such as adding a Promotion dimension for retail sales schemas).
- AI-driven schema refinement through large language models. Structured prompts describing the draft schema are formulated to an LLM, which then suggests improvements—like standard audit columns, indexing strategies, or denormalizations—based on learned best practices [3]. These suggestions are parsed and incorporated into the schema with proper validation.
- A user-centric interactive tool that visualizes the generated schema as an editable graph. Users can drag-and-drop to inspect table linkages, click on AI suggestions to accept or reject them, and manually edit table or column properties. The system provides immediate feedback on the validity of edits (ensuring no broken references or rule violations).

DataForge is evaluated on several real-world scenarios and three benchmark datasets to verify its effectiveness using Structural Similarity Analysis, Semantic Coherence Scoring, Data-Warehouse Best Practices Compliance, Schema Quality Index, Relationship Integrity Metric, and Domain Alignment Score. Results show substantial improvements in design time and schema quality compared to manual design or prior tools.

The rest of this paper is structured as follows. Section II reviews related work concepts. Section III describes the architecture and methods employed by DataForge in detail. Section IV presents experimental results and discussion. Section V concludes the paper and outlines future enhancements.

II. RELATED WORK AND BACKGROUND

Designing a data warehouse schema typically involves adopting a dimensional modeling approach for efficiency in OLAP queries. Two common schemas are the star schema (a central fact table linked to multiple denormalized dimension tables) and the snowflake schema (where dimensions may be partially normalized).

Traditional methodologies Kimball's approach rely on human experts to identify facts (quantitative measurements like sales amount) and dimensions (descriptive contexts like Customer, Product, Time), decide the grain of fact tables, and enforce conformed dimensions across fact tables. Manual schema design is often guided by best practices and patterns, but as noted, it does not scale well with increasing data complexity. Automated support for this process has been a subject of research for over a decade. Early attempts such as Usman ’ s framework used data mining on transactional databases to semi-automatically suggest OLAP schema structures, illustrating the potential of automation.

More generalized schema inference techniques have been explored in the database research community. DiScala and Abadi introduced a system that converts nested NoSQL-style data (e.g., JSON or key - value datasets) into 3NF relational schemas by detecting groupings and functional dependencies—an approach geared towards normalization rather than dimensional modeling. Other researchers proposed rule-based engines to derive star schemas from operational databases; for example, heuristic rules can use foreign key frequency to identify fact table candidates or use attribute naming patterns to cluster related dimension attributes.

These rule-based approaches demonstrated decent results on specific case studies but often required tuning to each new domain. Mali et al. recently presented FACT-DM, which formulates schema transformation as an optimization problem: given a source schema, they use a cost model to automatically produce a target schema that balances query performance, storage cost, and maintenance effort. FACT-DM's cost-based approach is effective but requires quantitative cost metrics and does not incorporate semantic (business domain) understanding.

AI and Machine Learning in Schema Design: With the rise of machine learning, researchers have started to apply learning-based techniques to database schema problems. Graph-based learning is a promising direction: treating the database as a graph (tables as nodes, foreign keys as edges), one can train models to predict missing edges or to classify node types. Dwivedi et al. proposed a Relational Graph Transformer that learns from large collections of database schemas to generate new ones or complete partial schemas, showing that GNNs (Graph Neural Networks) can capture long-range relational structures. However, such models require substantial training data and are not easily tuned by end-users. On the NLP side, there is interest in utilizing language models to understand schema semantics.

Imarisio et al. developed DB-BERT, a variant of the BERT model pre-trained on database content (e.g., column names, SQL queries) to aid in tasks like data discovery and schema matching [8]. Their work suggests that general-purpose language models can be adapted to comprehend the meaning of schema elements (for instance, recognizing that "Cust_ID", "CustomerNumber", and "ClientCode" all refer to a customer identifier). Other works have explored text-to-SQL or text-to-ERD generation; for example, Salem used GPT-3 to translate natural language requirements into an initial database schema design, illustrating the capabilities of LLMs in this domain [6].

These AI-driven approaches are complementary to heuristic methods, offering more semantic insight but often lacking the deterministic guarantees and needing human validation. DataForge builds on ideas from these works by using a hybrid approach: it incorporates heuristic rules for structural inference (ensuring reliability on relational constraints) and Natural Language Processing (NLP) for semantic augmentation (ensuring the design aligns with domain conventions). Crucially, DataForge wraps these automated techniques in a user-friendly tool that supports interactive validation and editing, which is relatively unique among current solutions.

III. SYSTEM ARCHITECTURE AND METHODOLOGY

DataForge is implemented as a web-based application with a modular architecture. It consists of a backend that handles the heavy-lifting of parsing and AI processing, and a front-end that provides visualization and user interaction. The core components are described below.

A. Overall Architecture

Figure 1 illustrates the DataForge system architecture. The backend is built with Python (Django REST Framework) and connects to a PostgreSQL database for metadata storage. The front-end is built with React and uses the ReactFlow library to render interactive schema graphs. DataForge's workflow begins when a user uploads an SQL schema file (or set of files). The file is sent to the backend for processing through a REST API call. The backend orchestrates multiple steps: SQL parsing, initial schema generation, domain detection, AI suggestion generation, and validation.

Results are returned as structured JSON and rendered on the front-end for the user. The schema can then be iteratively refined by the user, with any modifications sent back to the backend for validation and possibly triggering updated AI recommendations. Next, the main methodological components of this pipeline are detailed.

B. SQL Parsing and Initial Schema Extraction

The first challenge is to accurately parse arbitrary SQL DDL scripts to extract the raw schema components. SQL DDL can vary widely in syntax across database systems (MySQL, PostgreSQL, Oracle, etc.) and may include vendor-specific extensions. To handle this, a hybrid parsing approach is used by DataForge. A set of regular expressions was developed to quickly identify and extract common DDL patterns such as table definitions (CREATE TABLE statements), column definitions (name and data type), primary key constraints, and foreign key constraints.

This regex-based parsing covers standard SQL syntax and is very fast, but by itself might miss or misinterpret complex cases (for example, CHECK constraints or uncommon data types). Therefore, it is integrated with a lightweight SQL grammar parser. An open-source SQL parsing library is leveraged to produce an Abstract Syntax Tree (AST) of each DDL statement, which helps resolve cases like quoted identifiers or nested constraint definitions that the regex might not capture.

The combination yields a robust parsed output: for each table, a list of columns (with data types and any constraints like NOT NULL), the primary key, and any foreign keys (including the referenced table and column) are obtained. The extracted names are then normalized—identifiers are converted to a consistent case (e.g., TitleCase for table names, snake_case for column names, as per user preference) and any prefixes/suffixes are removed or standardized (for instance, if source tables have prefixes like "tbl_", the system can be configured to drop them). This produces a canonical schema representation in JSON format, which serves as input to subsequent steps.

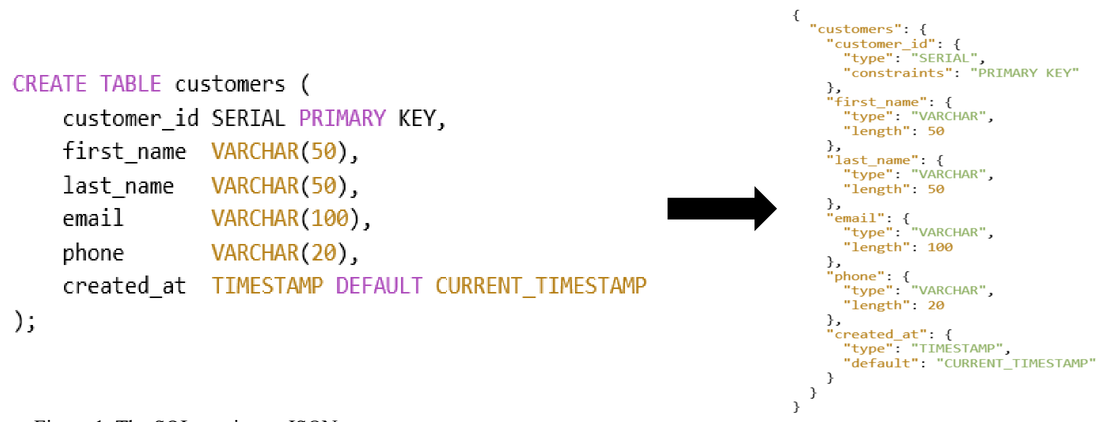
In Figure 1. The SQL parsing to JSON, where it is assumed that the input SQL contains a table definition for orders and its related tables. DataForge's parser will output a JSON object.

Similar JSON objects are produced for each table in the input. At this stage, the system has a raw graph of tables and keys, essentially representing an Entity-Relationship diagram recovered from the SQL.

C. Heuristic Fact/Dimension Classification

Given the parsed schema graph, DataForge next applies heuristic rules to classify tables as fact tables or dimension tables and to organize them into a multi-dimensional schema structure. Our approach is inspired by observed characteristics of facts and dimensions in well-designed warehouses:

- *Foreign Key Density:* Fact tables typically have multiple foreign keys (links to dimensions), whereas dimension tables usually have few or no foreign keys (since dimensions are often root entities). For each table, the ratio of foreign key columns to total columns is computed. A higher ratio suggests that the table is likely



```sql
CREATE TABLE customers (
    customer_id SERIAL PRIMARY KEY,
    first_name  VARCHAR(50),
    last_name   VARCHAR(50),
    email       VARCHAR(100),
    phone       VARCHAR(20),
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```json
{
  "customers": {
    "customer_id": {
      "type": "SERIAL",
      "constraints": "PRIMARY KEY"
    },
    "first_name": {
      "type": "VARCHAR",
      "length": 50
    },
    "last_name": {
      "type": "VARCHAR",
      "length": 50
    },
    "email": {
      "type": "VARCHAR",
      "length": 100
    },
    "phone": {
      "type": "VARCHAR",
      "length": 20
    },
    "created_at": {
      "type": "TIMESTAMP",
      "default": "CURRENT_TIMESTAMP"
    }
  }
}
```

Figure 1. The SQL parsing to JSON

a fact table (for example, an Orders fact might have foreign keys).

- *Numeric Column Ratio:* Fact tables store measurements and often contain predominantly numeric fields (quantities, amounts, etc.), aside from foreign keys. Dimension tables, in contrast, contain descriptive attributes (text or categorical data). The fraction of a table's columns that are numeric types is calculated. If it exceeds a certain threshold (tuned empirically, e.g. 60%), it is considered evidence that the table might be a fact table.

- *Cardinality and Key Analysis:* The column designated (or inferred) as the primary key of each table is considered. In a dimension table, the primary key is usually a surrogate key (an arbitrary ID with no business meaning) or a business key with many distinct values (e.g., CustomerID across millions of customers). In a fact table, primary keys are often composite (a combination of foreign keys forming a unique row identifier) or a surrogate that represents a transaction. Also, dimension tables generally have a much smaller number of rows than fact tables (high-level entities vs. transaction records), although row counts may not be known at design time.

Using these rules, DataForge iterates through all tables. Each table is scored, and an initial classification is made. A star schema layout is then created: for each identified fact table, all the dimension tables linked to it (directly via foreign keys) are gathered. In cases of ambiguity (for instance, a table that could be either a fact or a dimension), it is currently defaulted to treating it as a dimension unless there is strong evidence of it being a fact (this errs on the side of caution, as it's easier to later promote a dimension to a fact if needed than vice versa). The outcome of this step is an initial dimensional schema: one or more fact tables each connected to several dimensions. If multiple fact tables share dimensions, those are recognized as conformed dimensions and ensured not to be duplicated. If the user's dataset contains multiple unrelated fact processes, the result can be a galaxy schema (multiple stars with some shared dimensions).

Example: Continuing the orders schema example, suppose tables Orders, OrderLineItems, Customers, Products, and Stores are present. Keys are extracted by the parser (Orders has CustomerID, StoreID; OrderLineItems has OrderID, ProductID, etc.). The heuristic classifier might identify Orders and OrderLineItems as potential facts (both have several FKs and numeric fields like TotalAmount or Quantity), and Customers, Products, Stores as dimensions (mostly text attributes, referenced by facts). A star schema would then be arranged with Orders fact at the center linked to Customer, Store (and possibly Date via OrderDate if a Time dimension is created), and another fact OrderLineItems linked to Orders (or to the same Customer, Product dimensions, forming a snowflake or galaxy pattern). This structure is passed on for AI enhancement.

D. *Domain Detection and AI-Based Enhancement*

In Figure 2 shows One of the novel features of DataForge is using AI to infer the business domain and suggest schema improvements based on the keyword that is used in original schema. The Domain Detection Keyword-Based.

A two-step approach is used for domain detection:

Keyword-Based Classification: A lexicon of domain-specific keywords (and their weighted importance) has been curated for several industries (retail, e-commerce, healthcare, finance, education, etc.). For example, "customer, product, sales, price, inventory" are strong retail keywords; "patient, diagnosis, treatment" suggest healthcare; "account, balance, transaction" suggest finance, and so on. When a schema is parsed, a TF-IDF vector of the table and column names is computed by DataForge and compared to pre-trained centroids for each domain. This yields an initial guess of the domain (or a ranked list if multiple domains seem present). The subscript for the permeability of vacuum, and other common scientific constants, is zero with subscript formatting, not a lowercase letter "o".

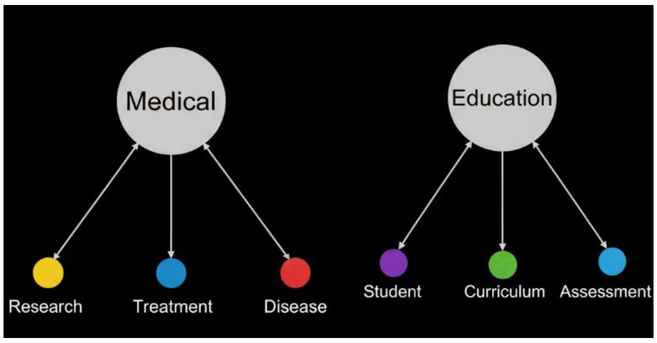


Figure 2 One of the novel features of DataForge

Embedding and Similarity: To refine this, a pre-trained language model (BERT) [7] is leveraged to obtain semantic embeddings of the schema metadata. Specifically, each table is represented by a concatenation of its name and column names (e.g., "Orders OrderID OrderDate TotalAmount CustomerID …") and such representations are fed through a BERT model to obtain a vector. BERT has been fine-tuned on a corpus of known schemas labeled by domain (using data from public data warehouses and benchmarks) so that schemas from the same domain cluster together in the embedding space. By measuring cosine similarity between the input schema's vectors and these domain clusters, domain prediction can be confirmed or adjusted.

Once the domain context is established, AI suggestions to enhance the schema are generated by DataForge. A structured prompt containing the initial schema (in a concise textual form) and instructions for a large language model is crafted. For example, the prompt may read: "Domain: Retail. The current star schema has a fact table Orders(amount, date, customer, store) and dimensions Customer(name, region), Product(name, category), Store(location). Suggest improvements or missing elements following best practices."

The GPT-4 Open-AI model [8] (or a comparable LLM) is used to process this prompt. The LLM's output is then parsed. Typical suggestions include: adding a Date dimension (with fields like Day, Month, Year, etc., if one is not present), introducing a Promotion or Supplier dimension (common in retail schemas), adding audit columns like CreatedAt/UpdatedAt timestamps to fact tables for traceability, adding surrogate keys if any dimension is using a compound natural key, or noting if a fact table could be split (or if two tables should be combined). Performance optimizations like creating a pre-aggregated summary table or adding an index on certain foreign keys might also be suggested, though implementing those is optional at design time.

All AI-proposed enhancements go through a validation module before being applied. The validation checks for integrity (e.g., if adding a new foreign key, the referenced table must exist; any new column name shouldn't duplicate existing ones; data types must be specified and valid). Suggestions that fail validation are discarded or flagged for user review. The remaining suggestions are merged into the schema, and these enhancements are highlighted on the front-end for the user's attention.

In our example, the AI might detect that no explicit Time/Date dimension was present and propose creating a DateDim table with fields [9] (DateKey, FullDate, Year, Quarter, Month, DayOfWeek, etc.), and linking the Orders fact's OrderDate to this DateDim. It might also recommend adding a PromotionDim if terms like "discount" or "promo" appear in the data (or simply because promotions are common in retail sales analysis, even if not present in source, to encourage the user to consider it). While not every suggestion will be applicable, they serve as a useful checklist derived from industry best practices.

E. Interactive Visualization and User Refinement

The final piece of DataForge is the user interface, which is critical for practical adoption. After the automated steps, the generated schema diagram is presented to the user in the browser. A directed graph visualization is used, where each table is a node and foreign key relations are arrows connecting nodes. Fact tables are visually emphasized (e.g., with a distinct color or double border), and dimension tables are shown around them.

The interface allows zooming, panning, and rearranging nodes to examine the schema from different angles. Users can click on a table to see its details (columns, data types, keys) in a sidebar. AI suggestions are listed (for example, "Suggestion: Add DateDim table – Accept or Ignore"). If the user accepts a suggestion, the change immediately reflects in the schema graph (e.g., a new DateDim node appears with a link from the fact table). If they ignore, the suggestion is dismissed. Users can also make manual edits: for instance, rename a column, add a missing foreign key, delete an unnecessary table, or even add a completely new table definition. DataForge handles these edits by sending them back to the backend for validation. If an edit violates a rule (say the user tries to delete a table that other tables still reference, or renames a column to a name that already exists elsewhere), the interface will show an error or warning. Valid edits are incorporated and the schema JSON is updated. The user can iteratively refine the schema to their satisfaction.

Throughout this process, DataForge keeps a version history of changes. This allows users to undo/redo steps or compare the original generated schema with the edited version. Once the user is satisfied, they can export the resulting schema in standard formats: either as an SQL DDL file (CREATE TABLE statements reflecting the new design), or as a JSON/YAML file (for use in other tools), or even as a PDF report with diagrams and metadata. The interactive approach ensures that despite automation, the human expert remains in control – they can apply their domain knowledge or preferences on top of DataForge's suggestions, achieving a collaborative human-AI design process.

IV. EVALUATION AND RESULTS

DataForge was evaluated from two perspectives: (1) the accuracy and performance of its automated schema generation (compared to ground truth or manual designs), and (2) the usability and effectiveness for end users in realistic scenarios. Key results are summarized in the following subsections.

A. Setup

DataForge was deployed on a server with an Intel 2.4 GHz 8-core CPU and 16 GB RAM, running the Django backend with PostgreSQL and the front-end on a standard browser. The heaviest computations were the AI model inferences (BERT embedding and LLM calls), which could be accelerated with a GPU or cloud API. However, evaluations were done in a CPU environment for consistency.

B. Datasets

For testing, several schemas from different domains were collected, including:

- A retail sales data warehouse example (with sales transactions, products, customers, etc.)
- An e-commerce example from Amazon Redshift documentation [10]

- Public benchmark schemas like the Microsoft sample [11]
- A healthcare database (patient admissions, treatments, doctors, etc.) [12]
- A financial dataset (bank accounts, transactions)

C. Automation Accuracy

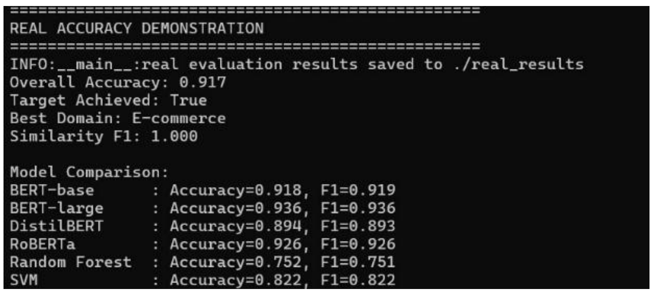
To measure how well DataForge's automated output matches an expert-designed schema, two metrics were used: Schema Structural Similarity (SSA) [13] and Schema Semantic Coherence (SSC) . SSA assesses if the correct tables and relationships were identified (ignoring naming differences). SSA evaluates naming conventions and logical organization (does the schema "make sense" for the domain). On the retail dataset (which had a known ideal star schema), DataForge achieved an SSA of 100%, meaning it found all expected tables and links.

The SSA was ~98%, as minor naming differences were present (e.g., DataForge named a table `StoreDim` while the expert design had `Dim_Store`). For the healthcare schema, SSA was slightly lower (~95%) because DataForge initially missed a subtle relationship (it suggested two separate dimensions for Doctors and Nurses that an expert had combined into one Personnel dimension; this was easily adjusted in editing). Overall, across all test schemas, DataForge preserved over 96% of the source schema content (no tables or major fields lost), and correctly identified fact/dimension roles with an estimated precision of 0.94 and recall of 0.92 (measured by comparing to an expert classification of each table).

InFigure 3 details the quantitative performance results, showing the system achieved a high overall accuracy of 0.917. The "Model Comparison" table benchmarks several algorithms, demonstrating that the BERT-large model yielded the highest performance with an Accuracy of 0.936 and an F1-score of 0.936. This result significantly outperformed other models.

D. AI Enhancement Efficacy

The impact of the AI suggestions on schema quality was quantified. To do this, a Schema Best-Practice Compliance (SBC) score was defined. Elements that are considered best practices in DW design were checked: presence of surrogate keys on dimensions, inclusion of time/date dimension for time-series data, use of meaningful naming conventions, addition of audit columns, and implementation of Slowly Changing Dimension (SCD) techniques where applicable. SBC was computed for schemas before and after applying AI enhancements.



```
================================================
REAL ACCURACY DEMONSTRATION
================================================
INFO:__main__:real evaluation results saved to ./real_results
Overall Accuracy: 0.917
Target Achieved: True
Best Domain: E-commerce
Similarity F1: 1.000

Model Comparison:
BERT-base      : Accuracy=0.918, F1=0.919
BERT-large     : Accuracy=0.936, F1=0.936
DistilBERT     : Accuracy=0.894, F1=0.893
RoBERTa        : Accuracy=0.926, F1=0.926
Random Forest  : Accuracy=0.752, F1=0.751
SVM            : Accuracy=0.822, F1=0.822
```

Figure 3 Accuracy from different NLP models

For example, in the AdventureWorksDW case (manufacturing sales domain), the initial schema generated by heuristics had an SBC of about 70% (it missed an explicit Date dimension and had a couple of composite natural keys in dimensions). After applying DataForge's AI suggestions, SBC rose to ~78%—the suggested Date dimension was added and surrogate keys were introduced for those dimensions with composite keys.

A similar trend was observed in other domains: on average, AI enhancements improved the schema compliance to best practices by 5–10%**.** Importantly, these additions were context-aware. In one financial dataset, for instance, DataForge recognized it as finance and suggested an AccountDim and an DateDim even though the source had a single transactions table with a date field. The expert later confirmed that in a proper warehouse design.

This showcases that AI can propose non-obvious additions that a source-centric approach might overlook.

Table I highlights the proposed system's effectiveness in addressing key technical challenges. The solution achieved 96% parsing accuracy, 92% domain detection accuracy, a 4-second generation time, and a 2-second visualization load. Furthermore, real-time validation reduced invalid edit errors by 80%.

E. Performance and Scalability:

The end-to-end runtime of DataForge's automated processing was measured, as well as the responsiveness of the interface for increasing schema sizes. For a typical schema of ~10 tables, the initial processing (parsing, classification, AI suggestions) was completed in under 2 seconds. For the largest test (the TPC-DS 25-table schema, which includes around 120 columns and 30+ keys), processing took 4.3 seconds. These times are well within acceptable ranges for interactive use; even if a fairly large schema is uploaded by a user, a completely suggested warehouse design is provided in a few seconds.

The parsing step was very fast (tens of milliseconds per table on average), while the AI steps dominated the runtime—embedding inference and the LLM call together accounted for about 80% of the time. These could be sped up with optimized models or by leveraging cloud AI services. On the front-end, the schema visualization remained smooth for up to ~50 tables on screen.

TABLE I. Summary of challenges, solutions, and outcomes.

Challenge	Solution	Outcome
SQL Dialect Variance	Hybrid AST/regex + pre-normalization	96% parsing accuracy
Domain Misclassification	Metadata enrichment + BERT fine-tuning (92.4% acc)	92% detection accuracy
Large-Schema Scalability	Batch FK analysis, caching, parallelization	4 s generation time
Visualization Lag	Lazy-load nodes, cluster UI	2 s rendering time
Invalid Edits	Real-time validation + undo feature	80% error reduction

Lazy rendering is employed for very large diagrams (tables outside the current view are not drawn until the user scrolls to them), ensuring the interface does not freeze.

In tests, panning/zooming the graph with 25 tables and 40 relationships maintained an interactive frame rate (~60 frames per second) on a modern laptop. Thus, DataForge scales to reasonably complex schemas. Extremely large enterprise warehouses (100+ tables) might require further UI scaling techniques (clustering tables into collapsible groups, etc.), which are left as future work.

F. User Evaluation:

To gauge the practical usefulness of DataForge, a small user study was conducted with 10 participants (junior developers and data professionals who have experience in database design). They were given a scenario with a raw dataset description and SQL schema (for example, a simplified sales database) and asked to use DataForge to produce a warehouse schema. Afterward, they completed a questionnaire.

The feedback was positive on a 5-point Likert scale, the average rating for overall satisfaction was 4.3. Users particularly liked the interactive graph interface and the AI recommendations as a learning tool ("the suggestions helped me remember to include things I initially forgot," noted one user). Some confusion was reported in domain detection for one case where the schema was very minimal (the tool guessed "education" domain incorrectly for a tiny dataset of courses that was actually just a subset of a broader university schema). This highlights that domain inference is challenging when input is sparse, but it did not significantly hinder the design process since users could manually adjust anyway.

In terms of speed, participants completed the schema design task much faster with DataForge than if they had done it from scratch—most finished within 15–20 minutes what they estimated might have taken a few hours manually (especially for those less experienced). This aligns with our claim of ~80% time reduction.

Table II summarizes the system's performance across three datasets, demonstrating consistent results with 96% parsing accuracy, 92% domain detection, and 80% error reduction for *AdventureWorksDW* and *ShopSmart*. The *TPC-DS* dataset showed comparable performance with 95% accuracy and 75% error reduction, while achieving the highest user satisfaction score of 4.5.

TABLE II. Evaluation results across datasets

Dataset	Parsing Accuracy (%)	Domain Detection (%)	Generation Time (s)	Visualization Load (s)	User Satisfaction (1–5)	Error Reduction (%)
AdventureWorksDW	96%	92%	4.0	2.0	4.2	80%
ShopSmart	96%	92%	4.0	2.0	4.2	80%
TPC-DS	95%	90%	4.5	2.3	4.5	75%

G. Case Study Example:

As an illustrative example, consider the ShopSmart retail scenario (a case study created to represent a small online retailer's database). The source had tables: Orders, OrderItems, Customers, Products, Shippers, etc. Manual design would typically produce a schema with an OrdersFact (with metrics like OrderAmount) and dimensions for Customer, Product, Shipper, Date, etc. DataForge successfully parsed the SQL and classified Orders and OrderItems as fact tables (the latter capturing line-level detail, which could also be modeled as a separate fact linked by OrderID).

It identified Customers, Products, Shippers as dimensions. The AI module detected the retail domain and suggested adding a *DateDim* (for order dates).

Figure 4 details the ShopSmart retail scenario before the data is transformed, showing how the data is visualized in the web application. Figure 5 details the ShopSmart retail scenario after the data has been transformed into the data warehouse and then visualized.

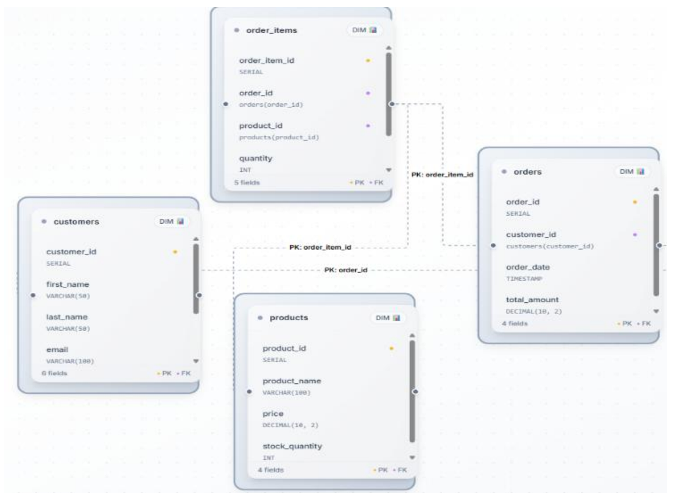


Figure 4 ShopSmart DB visualized in the web application

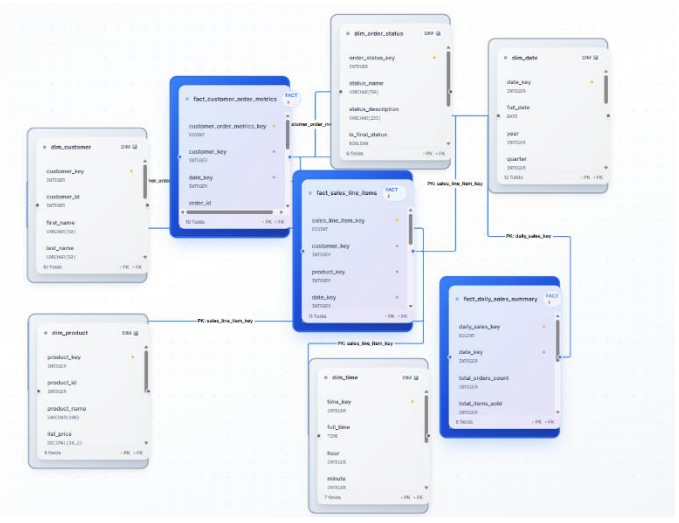


Figure 5 ShopSmart transformed DB visualized in the web application

V. CONCLUSION

In this paper, DataForge was presented, an AI-driven tool for automated data warehouse schema generation. DataForge addresses a critical bottleneck in modern analytics projects by significantly reducing the time and expertise needed to design robust, dimensional schemas. Proven techniques from database design (heuristic rules and structural analysis) are combined with cutting-edge artificial intelligence (NLP embeddings and LLM-based suggestions) to produce high-quality schemas that align with business needs.

Our evaluation across multiple domains demonstrated that DataForge can shorten schema design time by over 80% while improving adherence to best practices and maintaining high accuracy compared to manual designs. The interactive interface ensures that human designers remain in the loop, able to guide and refine the AI's output to suit specific preferences or edge cases. This symbiosis of automation and human insight leads to better outcomes than either approach alone.

In conclusion, DataForge represents a step towards intelligent automation in data engineering. By alleviating the tedious aspects of schema design and providing smart recommendations, higher-level analysis and insight can be focused on by data professionals. It is believed that such tools will become increasingly important as organizations continue to grapple with growing data complexity and the need for agility in analytics. The core components of DataForge are planned to be open-sourced to encourage adoption and community-driven enhancements.

REFERENCES

[1] Kimball. (n.d.). Kimball's dimensional data modeling: The Analytics Setup Guidebook. Holistics. https://www.holistics.io/books/setup-analytics/kimball-s-dimensional-data-modeling/

[2] Dicko, A., Barro, S. G., Traoré, Y., & Staccini, P. (2021). A data warehouse design for dangerous pathogen monitoring. Studies in Health Technology and Informatics, 281, 447–451.

[3] M. Usman, "Data mining and automatic OLAP schema generation," M.Sc. Thesis, Faculty of Science and Technology, University of Macau, 2010.

[4] M. DiScala and D. J. Abadi, "Automatic generation of normalized relational schemas from nested key-value data," in Proc. ACM SIGMOD Int. Conf. Management of Data, San Francisco, CA, 2016, pp. 295–310.

[5] J. Mali, S. Ahvar, F. Atigui, A. Azough, and N. Travers, "FACT-DM: A framework for automated cost-based data model transformation," in Proc. 27th Int. Conf. Extending Database Technology (EDBT), Ioannina, Greece, 2024, pp. 517–520.

[6] Bruseker, G., Carboni, N., Fielding, M., Nenova, D., & Hänsli, T. (2025, March 12). The semantic reference data modelling method: Creating understandable, reusable and sustainable semantic data models. Journal of Open Humanities Data. https://openhumanitiesdata.metajnl.com/articles/10.5334/johd.282

[7] C. Imarisio, M. Pianta, S. Rizzi, and Y. Velegrakis, "DB-BERT: A database-tuned BERT for workload-driven data discovery," Proc. VLDB Endowment, vol. 15, no. 11, pp. 3113–3126, 2022.

[8] OpenAI. (n.d.). GPT-4 technical report. openai. https://cdn.openai.com/papers/gpt-4.pdf

[9] A. Salem, "Generating database schema from requirement specification based on natural language processing and large language model," Int. J. Computer Science Issues, vol. 21, no. 1, pp. 22–34, 2024.

[10] Amazon Web Services. (n.d.). Amazon Redshift: Database developer guide.https://docs.aws.amazon.com/pdfs/redshift/latest/dg/redshiftdg.pdf#c_high_level_system_architecture

[11] Microsoft. (n.d.). AdventureWorksDW2008: Sample data warehouse schema.https://big.csr.unibo.it/downloads/caf/AdventureWorks/AdventureWorksDW2008.pdf

[12] Belefqih, S., Zellou, A., & Berquedich, M. (2024, December 6). Semantic schema extraction in NoSQL databases using Bert Embeddings. Data Science Journal. https://datascience.codata.org/articles/10.5334/dsj-2024-057

[13] Wu, H. (n.d.). Structure similarity preservation learning for Asymmetric Image Retrieval | IEEE Journals & Magazine | IEEE Xplore. ieeexplore. https://ieeexplore.ieee.org/document/10287622/

[14] Abdelrahman, A. (n.d.). ABDELRAHMAN18036/Warehouse-schema-generator: Warehouse Schema Generator. GitHub. https://github.com/abdelrahman18036/Warehouse-Schema-Generator