Automatic Data Transformation Using Large Language Model – An Experimental Study on Building Energy Data

Ankita Sharma[a,b]*, Xuanmao Li[b,g]*, Hong Guan[a,b]*, Guoxin Sun[b]*, Liang Zhang[c,f]‡, Lanjun Wang[d],
Kesheng Wu[a], Lei Cao[c], Erkang Zhu[e], Alexander Sim[a], Teresa Wu[b], Jia Zou[a,b]†

Lawrence Berkeley National Lab[a], Arizona State University[b], University of Arizona[c], Tianjin University[d]
Microsoft[e], National Renewable Energy Laboratory[f], Huazhong Univ of Science and Technology[g]

{ashar236, xuanmaol, hguan6, gsun20}@asu.edu, liangzhang1@arizona.edu, wang.lanjun@outlook.com,
kwu@lbl.gov, caolei@arizona.edu, erkang.zhu@microsoft.com, asim@lbl.gov, {teresa.wu, jia.zou}@asu.edu

Abstract—**Existing approaches to automatic data transformation are insufficient to meet the requirements in many real-world scenarios, such as the building sector. First, there is no convenient interface for domain experts to provide domain knowledge easily. Second, they require significant training data collection overheads. Third, the accuracy suffers from complicated schema changes. To address these shortcomings, we present a novel approach that leverages the unique capabilities of large language models (LLMs) in coding, complex reasoning, and zero-shot learning to generate SQL code that transforms the source datasets into the target datasets. We demonstrate the viability of this approach by designing an LLM-based framework, termed `SQLMorpher`, which comprises a prompt generator that integrates the initial prompt with optional domain knowledge and historical patterns in external databases. It also implements an iterative prompt optimization mechanism that automatically improves the prompt based on flaw detection. The key contributions of this work include (1) pioneering an end-to-end LLM-based solution for data transformation, (2) developing a benchmark dataset of 105 real-world building energy data transformation problems, and (3) conducting an extensive empirical evaluation where our approach achieved 96% accuracy in all 105 problems. `SQLMorpher` demonstrates the effectiveness of utilizing LLMs in complex, domain-specific challenges, highlighting the potential of their potential to drive sustainable solutions.**

Index Terms—**large language model, data transformation, smart building, ChatGPT, Text2SQL**

I. INTRODUCTION

A recent study [1] showed that in 2022, the end-use energy consumption by the building sector accounted for **40%** of total US energy consumption. It indicates that the energy management of buildings plays an important role in meeting the goals of energy sustainability [2]. Automatic building energy management, including design, certification, compliance, real-time control, operation, and policy-making, requires the integration of data from diverse sources in both the private and public sectors. Harmonizing these data, as illustrated in Fig. 1, remains a manual process. Extensive labor and expertise are thus required throughout the data lifecycle in the building sector. However, the state-of-the-art data transformation tools, such as Auto-Transform [3], Auto-Pipeline [4], and Auto-Tables [5], are not effective due to the following gaps:

---

* These authors made equal contributions; †Jia Zou is the corresponding author; ‡Liang Zhang is the contact for the datasets and use cases.

- These tools are not publicly available and are based on supervised learning approaches, requiring non-trivial data labeling and training overheads.
- The data transformation logic in the building sector involves multiple combinations of aggregation, attribute flattening, merging, pivoting, and renaming relationships between the source and the target. They are more complicated than existing data transformation benchmarks [4], [5]. In addition, the accuracy achieved by the state-of-art tools on these simpler benchmarks is below 80% [4], [5], indicating it still requires human efforts to fix a significant portion of the cases.
- Converting a building dataset to a target schema requires domain knowledge about both source/target schemas, which are available in domain-specific knowledge as illustrated in Fig. 4 and Fig. 6. However, there is no easy way to directly supply such knowledge in existing data transformation tools.

To close the gaps, we once considered fine-tuning a pre-trained transformer model like BERT [6] to directly transform source data to target data [7]. However, we identified many shortcomings of this approach. First, it is hard to formulate one unified predictive problem to transform data for all types of schema changes. Second, the transformation process is slow to handle large-scale data. Third, preparing a fine-tuning dataset for each task could also be challenging.

This work proposes a novel and better approach, termed `SQLMorpher`, which solves the problem in two steps. The first step is formulated as a Text2SQL problem [8]–[11]. We apply the LLM model to *generate Structured Query Language (SQL) code that converts a source dataset into a target dataset*. This step focuses on schema mapping, so we do not need to upload the entire source datasets. The second step applies the generated SQL code to efficiently transform the entire dataset in relational databases.

Our approach is motivated by several key observations: (1) **LLMs demonstrate superior performance in complex reasoning tasks**. In the building sector, domain experts often document the semantics of the source and the target tables in natural language. LLMs can better understand such descriptions and reason about the relationships between the source and target than smaller pre-trained models. (2) **LLMs have demonstrated strong coding and code explanation**

(a) Transformation based on Group-By, Aggregation, and Pivoting

(b) Transformation based on Attribute Group and Merge

(c) Transformation based on Attribute Merge and Attribute Name Change
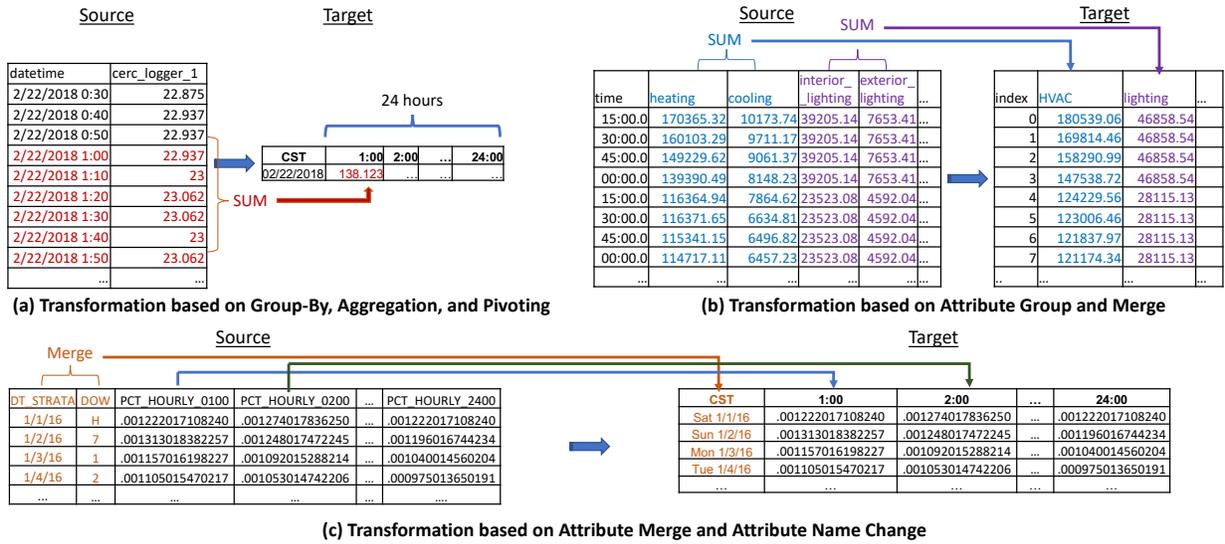
Fig. 1. Building electricity usage profiles are captured in a variety of data formats and need to be converted in order to be used together. This figure provides three simplified examples.

capability [12]. In addition, SQL's declarative nature makes it easier to map data transformation queries in natural language to SQL queries. (3) **LLMs have outstanding capabilities in zero-shot and few-shot adaptation and generalization.** Therefore, none or only a few training examples are needed.

Existing Text2SQL works [8]–[11] focus on read-only tasks, but cannot handle creation and modification tasks. To effectively handle the data transformation tasks with LLM, we need to add a number of challenges:

• **Schema Change Challenge:** Different from existing Text2SQL works, `SQLMorpher` needs to generate the query that maps data from the source schema to the target schema.

• **Prompt Engineering Challenge:** Designing a unified prompt to handle different types of schema changes and data transformation contexts is boring and tedious.

• **Accuracy Challenge:** Most importantly, the code generated by LLMs could be error-prone and even dangerous (e.g., leading to security concerns such as SQL injection attacks).

To address these challenges, the proposed system, as illustrated in Fig. 2, consists of the following unique components:

First, **a prompt generator** is designed to provide a unified prompt template. It allows external tools to be easily plugged into the component, such as domain-specific databases, vector databases that index historical successful prompts, and existing schema change detection tools [13]–[15] to retrieve various optional information. The prompt generator reduces the prompt size by using a few sample data records to replace the source datasets for generating the SQL code applicable to transforming the entire source datasets.

Second, **an automatic and iterative prompt optimization component** executes the SQL code extracted from the LLM response in a sandbox database that is separated from user data. It also automatically detects flaws in the last prompt and adds a request to fix the flaws in the new prompt. Examples of the flaws include errors appearing in the last LLM response, the errors that occurred when executing the SQL query generated by the LLM, as well as insights extracted from these errors based on rules.

Our Key Contributions are summarized as follows:

• We are the first to apply LLMs to generate SQL code for data transformation. Our system, termed `SQLMorpher`, includes a prompt generator that can be easily integrated with domain-specific knowledge, high-level schema-change hints, and historical prompts. It also includes an iterative prompt optimization tool that identifies flaws in the prompt for enhancement. We implemented an evaluation framework based on `SQLMorpher`. (See details in Sec. III)

• We set up a benchmark that consists of 105 real-world data transformation cases in 15 groups in the smart building domain. We document each case using the source schema, the source data examples, the target schema, available domain-specific knowledge, the schema hints, and a working transformation SQL query for users to validate the solutions. We have made the benchmark publicly available to benefit multiple communities including smart building, Text2SQL, and automatic data transformation [1] [2]. (See details in Sec. IV-B)

• We have conducted a detailed empirical evaluation with ablation studies. `SQLMorpher` using ChatGPT-3.5-turbo-16K achieved up to **96%** accuracy in **105** real-world cases in the smart building domain. We verified that our approach can generalize to scenarios beyond building energy data, such as COVID-19 data and existing data transformation benchmarks. We also managed to compare `SQLMorpher` to state-of-the-art data transformation tools such as Auto-Pipeline (though these tools are not publicly available) on their commercial benchmark. The results showed that `SQLMorpher` can achieve **81%** without using any domain knowledge, and **94%** accuracy using domain knowledge, both of which outperform Auto-Pipeline's

[1] https://github.com/asu-cactus/Data_Transformation_Benchmark
[2] https://github.com/asu-cactus/ChatGPTwithSQLscript

accuracy on this benchmark. We also summarized a list of insights and observations that are helpful to communities. (See details in Sec. IV)

II. RELATED WORKS

Existing Text2SQL tools [8]–[11] automatically generate SQL code to answer text-based questions on relations. However, existing Text2SQL tools focus on generating selection queries. According to our knowledge, there do not exist any Text2SQL tools that support modification queries (e.g., insertions) that are required by data transformation. In addition, we surveyed multiple Text2SQL benchmarks including Spider [16], SQUALL [17], Criteria2SQL [18], KaggleD-BQA [19], and so on. However, we didn't find any data transformation use cases in these benchmarks, which also indicates that data transformation problems are not the focus of today's Text2SQL research.

Existing automatic data transformation [3]–[5], [20]–[28] fall in two categories: (1) Transform-by-Example (TBE) [20]–[27] infers transformation programs based on user-provided input/output examples, which have been incorporated into popular software such as Microsoft Excel, Power BI [29], and Trifacta [30]. However, these works require users to provide examples of the transformed tuples, which is challenging for complicated data transformations. (2) To address the issue, Transform-by-Target (TBT) [3]–[5] is recently proposed. Works in this category, such as Auto-Transform [3], Auto-Pipeline [4], and Auto-Tables [5], transform data based only on input/output data schemas and optionally output data patterns. As mentioned, they learn a pipeline of data transformation operators using deep learning. They cannot easily integrate domain-specific knowledge represented in natural language or other formats. Although those tools are not publicly available, we conducted a comparison by running our approach on their benchmark, as detailed in Sec. IV-F.

III. SQLMorpher SYSTEM DESIGN

As illustrated in Fig. 2, the SQLMorpher system consists of a prompt generator, a large language model (LLM), a SQL execution engine, and a component for iterative prompt optimization. In this section, we describe each component in detail. Although SQLMorpher is primarily engineered to evaluate the LLM in our target use scenarios, it is a first-of-a-kind design that has research values in defining the workflows and the interfaces between LLM and external tooling for the unique data transformation problem.

A. Prompt Generation

We designed a prompt template as illustrated in Fig. 3. The naive user must provide minimal information, such as the source and target table schemas and examples of the tuples in the source dataset. Although a source table could contain many tuples, SQLMorpher only demonstrates to the LLM a few examples, which are sufficient to generate code for correctly transforming the whole table. Despite the sampling techniques that can be applied here, we chose to randomly sample 5 source tuples in the evaluation. If source tuples are not available, we asked the LLM to generate 5 source tuples.

All other information is optional but is helpful for complicated transformation cases. We designed the prompt generator to retrieve additional information from external databases easily. Such information includes:

(1) **Domain-specific information**, which explains the semantics of each attribute in the source table and the target table. Given LLMs' diverse and ocean-volume training corpus, such explanations are not required for many domains, and so it is marked as optional. However, we found that using domain knowledge to enhance the prompt could be critical for many smart building data transformation cases. This information can be retrieved from a domain-specific database, as illustrated in Fig. 4. In this example, basic information plus the domain-specific information that explains only the target table is sufficient to address the first example, as shown in Fig. 1a.

(2) **Schema change hints** suggests how the source schema is mapped to the target schema. Given the strong semantic reasoning capability of LLMs, hints are also optional. We found that some high-level hints, such as "use aggregation", are sufficient for LLMs to generate correct Group-By clause and aggregation functions in most scenarios. As illustrated in the brown box in Fig. 5, such information can be provided by (a) a rule engine that analyzes domain-specific databases as illustrated in Fig. 6, (b) a schema mapping tool such as Starmie [13], or (c) even an LLM itself (e.g., using a separate LLM prompt that asks the LLM to identify schema changes between the source and the target). Fig. 6 illustrates the example information that is available in a domain-specific database for smart buildings that can be leveraged to generate schema change hints. In this experimental study, most schema change hints are derived from the domain-specific databases as illustrated in Fig. 4 and Fig. 6.

(3) **Demonstrations** add a few examples of historical prompt-response pairs to the prompt to perform few-shot learning. It is critical that the demonstrating prompts need to be similar to the current prompt. For SQLMorpher design, we choose to store the embedding vectors of historically successful prompts in a vector database, such as Faiss for top-k nearest-neighbor search, as illustrated in Fig. 5. In this example, the prompt will fail unless it includes both the schema change hints (in the brown box) and the demonstration (in the purple box). We used the ChatGPT 3.5-turbo-16k model API in August 2023 to generate all examples in this section.

To retrieve the various types of information to augment the prompt, the SQLMorpher design includes a callback system. Each type of information corresponds to an event, and the user can register one or more callback functions with an event. Each callback function is expected to return a JSON object that specifies the retrieved information as well as a status code and error message that specifies connection or execution errors, if any. When generating a prompt, SQLMorpher will go through all types of information, and invoke all callback functions associated with each information type.
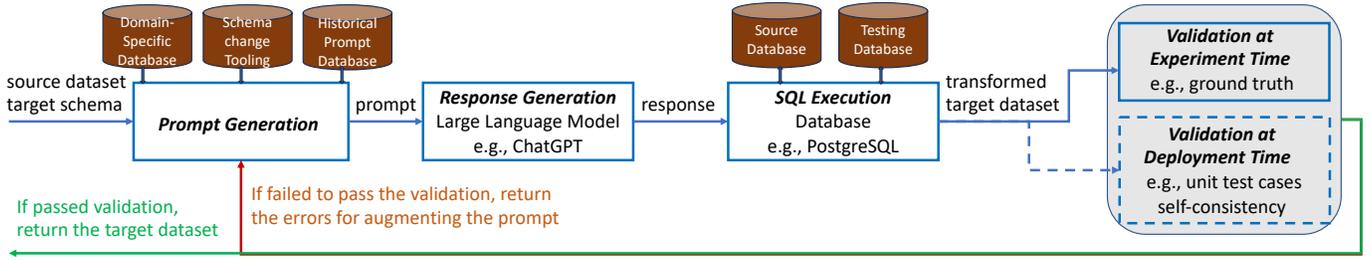
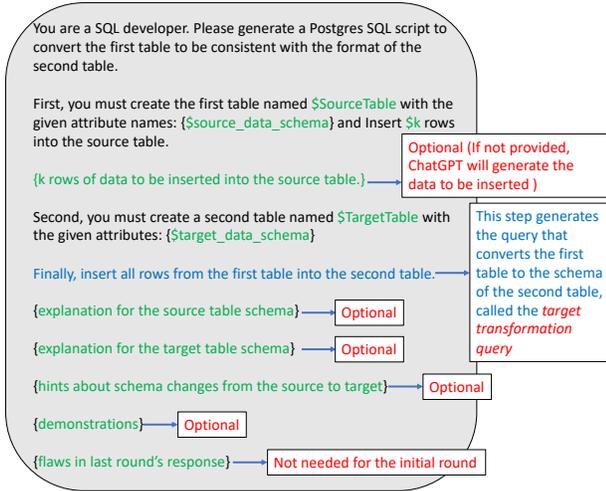Fig. 2. `SQLMorpher`: Automatic Data Transformation based on LLM

Fig. 3. Prompt Template for single-table transformation.

B. SQL Execution

Compared to existing Text2SQL that focuses on selection queries that are read-only, leveraging LLM to generate modification queries is more complicated, partially because running the generated query may raise security concerns. In the initial iteration for a given user request, the system automatically duplicates the source dataset in a separate PostgreSQL database that serves as a sandbox environment to isolate the errors, if the duplicate does not exist. This is to ensure that the generated code will not corrupt the source dataset. Then, the script creates the target table. Finally, it runs the generated query to transform the entire source dataset into the target format and insert all transformed tuples into the target table. If another iteration is needed, e.g., the response cannot pass the validation tests, the target table will be removed or archived before running the next iteration.

C. Validation

The validation in the production environment could be challenging due to the lack of ground truth. It needs an automatic quality measurement (e.g., unit test cases, self-consistency, or accuracy of downstream tasks) for the transformed data, which we leave for future work to address.

In this work, we manually prepare the ground truth transformation queries for each transformation case in the experimental environment. At the validation stage, the ground truth transformation query will be executed against the source table, resulting in the target table, which is called the *ground truth target table*. At the same time, by executing the target transformation query contained in the LLM response, as described in Sec. III-B, we can also obtain a target table, which is called the *generated target table*.

We designed a validation script, which compares the *generated target table* to the *ground truth target table*. The comparison first validates whether two tables have the same number of attributes and tuples. Then, it performs attribute-reordering and tuple-sorting to ensure two tables share the same column-wise and row-wise orderings. Furthermore, the script will compare the similarity of the values for each attribute in the two tables. We use the ratio of the number of equivalent values (difference should be less than e^{-10}) to the total number of values to measure the similarity of numerical attributes. We use the Jaccard similarity to measure the similarity of categorical and text attributes. We average the similarity for each attribute to derive an overall similarity score. If the similarity score is below 1, the validation fails.

D. Iterative Prompt Optimization

This component is incorporated to evaluate the LLM's potential self-optimization capability for the data transformation problem. If the validation fails, the prompt will be automatically augmented by identifying errors in the prompt: (1) errors mentioned in the LLM response or met when executing the generated transformation query; (2) errors detected in the transformed dataset, e.g., reporting the difference between the schema of the transformed dataset and the target schema; (3) inconsistency between the schema change hint and the response query, e.g., reporting if the hint specifies to use aggregation, but no Group-By or aggregation functions have been used. Then, these errors will be appended to the prompt, and the new prompt will be sent back to the LLM, and it will repeat this process until it passes the validation, the maximum number of iterations has been reached, or the new prompt has no difference with the last prompt.

An example of *useful* prompt flaws that we observed in those cases is "ERROR: INSERT has more expressions than target columns LINE 100: PCT_HOURLY_2500". Before adding this error to the prompt, ChatGPT cannot correctly handle an attribute that exists in the source table but not in the target table, PCT_HOURLY_2500. Adding the error to the prompt will resolve the problem.

You are a SQL developer. Please generate a Postgres SQL script to convert the first table to be consistent with the format of the second table.
First, you must create the first table named source1 with the given attributes: (datetime, cerc_logger_1), and insert 10 rows into the first table:
2/22/2018 0:30, 22.875
2/22/2018 0:40, 22.937
2/22/2018 0:50, 22.937
2/22/2018 1:00, 22.937
2/22/2018 1:10, 23
2/22/2018 1:20, 23.062
2/22/2018 1:30, 23.062
2/22/2018 1:40, 23
2/22/2018 1:50, 23.062
Second, you must create a second table named target1 with only the given attributes: (CST, 1:00, 2:00, 3:00, 4:00, 5:00, 6:00, 7:00, 8:00, 9:00, 10:00, 11:00, 12:00, 13:00, 14:00, 15:00, 16:00, 17:00, 18:00, 19:00, 20:00, 21:00, 22:00, 23:00, 24:00)
Finally, insert all rows from the first table to the second table.

Response to the Basic Prompt

```
-- Insert data from source1 into target1
INSERT INTO target1 (CST, 1:00, 2:00, 3:00, 4:00, 5:00, 6:00, 7:00, 8:00, 9:00,
10:00, 11:00, 12:00, 13:00, 14:00, 15:00, 16:00, 17:00, 18:00, 19:00,
20:00, 21:00, 22:00, 23:00, 24:00)
SELECT
  DATE_TRUNC('day', datetime) AS CST,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 1 THEN cerc_logger_1 END) AS 1:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 2 THEN cerc_logger_1 END) AS 2:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 3 THEN cerc_logger_1 END) AS 3:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 4 THEN cerc_logger_1 END) AS 4:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 5 THEN cerc_logger_1 END) AS 5:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 6 THEN cerc_logger_1 END) AS 6:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 7 THEN cerc_logger_1 END) AS 7:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 8 THEN cerc_logger_1 END) AS 8:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 9 THEN cerc_logger_1 END) AS 9:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 10 THEN cerc_logger_1 END) AS 10:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 11 THEN cerc_logger_1 END) AS 11:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 12 THEN cerc_logger_1 END) AS 12:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 13 THEN cerc_logger_1 END) AS 13:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 14 THEN cerc_logger_1 END) AS 14:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 15 THEN cerc_logger_1 END) AS 15:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 16 THEN cerc_logger_1 END) AS 16:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 17 THEN cerc_logger_1 END) AS 17:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 18 THEN cerc_logger_1 END) AS 18:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 19 THEN cerc_logger_1 END) AS 19:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 20 THEN cerc_logger_1 END) AS 20:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 21 THEN cerc_logger_1 END) AS 21:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 22 THEN cerc_logger_1 END) AS 22:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 23 THEN cerc_logger_1 END) AS 23:00,
  MAX(CASE WHEN EXTRACT(HOUR FROM datetime) = 0 THEN cerc_logger_1 END) AS 24:00
FROM source1
GROUP BY DATE_TRUNC('day', datetime);
```

(a) Basic Prompt and its response has only one error in the aggregation function, which should be SUM rather than MAX

Prompt with Domain Knowledge

We append the below domain-specific explanation to the basic prompt:

Company	Table ID	Description
C1	C1_RHH	Residential high summer use; high winter use
C1	C1_RHL	Residential high summer use; low winter use
C1	C1_RHT	Residential-with heat, each hour each month
C1	C1_L	Total Load each hour each day

Note that the second table records the total load in each hour each day.

Domain-Specific Database
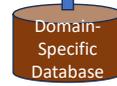
Response to the Prompt with Domain Knowledge

```
-- Insert data from source1 into target1
INSERT INTO target1 (CST, 1:00, 2:00, 3:00, 4:00, 5:00, 6:00, 7:00, 8:00, 9:00, 10:00, 11:00, 12:00, 13:00, 14:00,
15:00, 16:00, 17:00, 18:00, 19:00, 20:00, 21:00, 22:00, 23:00, 24:00)
SELECT
  DATE_TRUNC('day', datetime) AS CST,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 1 THEN cerc_logger_1 END), 0) AS 1:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 2 THEN cerc_logger_1 END), 0) AS 2:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 3 THEN cerc_logger_1 END), 0) AS 3:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 4 THEN cerc_logger_1 END), 0) AS 4:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 5 THEN cerc_logger_1 END), 0) AS 5:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 6 THEN cerc_logger_1 END), 0) AS 6:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 7 THEN cerc_logger_1 END), 0) AS 7:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 8 THEN cerc_logger_1 END), 0) AS 8:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 9 THEN cerc_logger_1 END), 0) AS 9:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 10 THEN cerc_logger_1 END), 0) AS 10:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 11 THEN cerc_logger_1 END), 0) AS 11:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 12 THEN cerc_logger_1 END), 0) AS 12:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 13 THEN cerc_logger_1 END), 0) AS 13:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 14 THEN cerc_logger_1 END), 0) AS 14:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 15 THEN cerc_logger_1 END), 0) AS 15:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 16 THEN cerc_logger_1 END), 0) AS 16:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 17 THEN cerc_logger_1 END), 0) AS 17:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 18 THEN cerc_logger_1 END), 0) AS 18:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 19 THEN cerc_logger_1 END), 0) AS 19:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 20 THEN cerc_logger_1 END), 0) AS 20:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 21 THEN cerc_logger_1 END), 0) AS 21:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 22 THEN cerc_logger_1 END), 0) AS 22:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 23 THEN cerc_logger_1 END), 0) AS 23:00,
  COALESCE(SUM(CASE WHEN EXTRACT(HOUR FROM datetime) = 0 THEN cerc_logger_1 END), 0) AS 24:00
FROM source1
GROUP BY DATE_TRUNC('day', datetime);
```

This script follows the structure of the previous scripts but adjusts the aggregation function to calculate the total load for each hour of the day. It sums up the cerc_logger_1 values within each hour and day to populate the corresponding columns in the target1 table. The COALESCE function is used to handle cases where there's no data for a particular hour.

(b) Basic Prompt with domain-specific explanation for the target table returns the correct target transformation query

Fig. 4. Prompt-Response for the example illustrated in Fig. 1a.

IV. EXPERIMENTAL EVALUATION

In this section, we first describe the goal of the comparison study and all baselines that were used. Then, we present a benchmark, which is the *first* benchmark for smart building data standardization problems. We further describe the setup of the experiments and the evaluation metrics. Ultimately, we will present and analyze the results and summarize key findings.

A. Comparison and Baselines

In this work, we mainly compare the effectiveness of six different types of initial prompt templates:

- Prompt-1: Basic prompt with domain-specific description for the target schema.
- Prompt-2: Prompt-1 with a domain-specific description for the source schema.
- Prompt-3: Prompt-2 with schema change hints.
- Prompt-1+Demo: Prompt-1 with one demonstration.
- Prompt-2+Demo: Prompt-2 with one demonstration.
- Prompt-3+Demo: Prompt-3 with one demonstration.

The first three prompt templates are designed for zero-shot learning when there does not exist a database of abundant historical working prompts. The last three prompt templates are designed for one-shot learning.

We also considered comparing our approach to Auto-Pipeline [4], which is a state-of-the-art automatic data transformation tool that only requires schema and tuple examples of the source and target tables and applies deep reinforcement learning to synthesize the transformation pipeline.

B. Benchmark Design

1) Building Energy Data Transformation: We collected 105 data transformation examples in the smart building domain from 21 energy companies in the United States. These examples are divided into 15 groups so that each group has one target dataset and multiple source datasets of different types. Each source needs to be converted to the target format in the group. The groups are described in Tab. I. In Tab. II, we further show more statistics of the 105 test cases by groups: (1) the
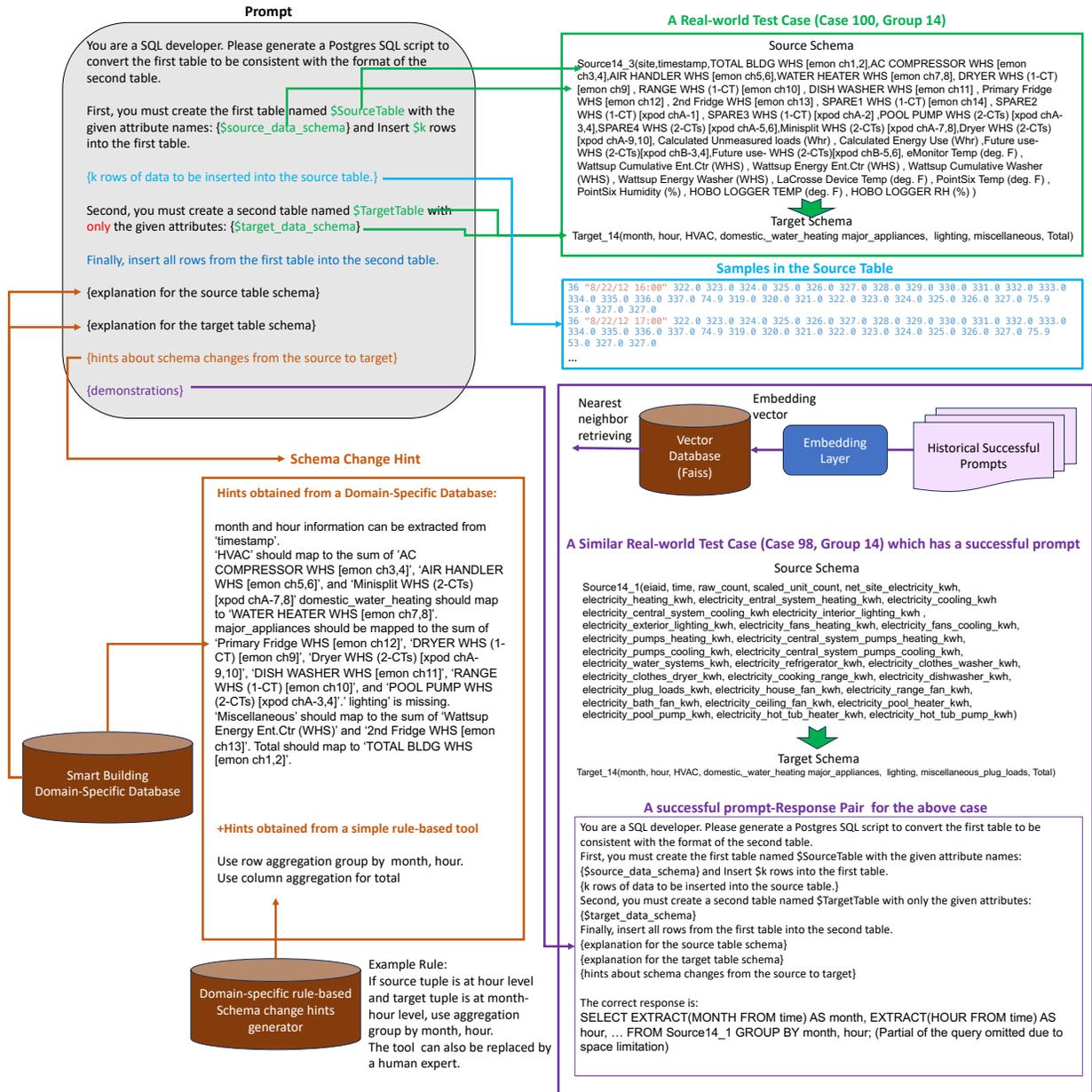
Fig. 5. A Working Prompt for a Real-World Case (Case 100 in Group 14 in Tab. I).

number of distinct SQL keywords used in the ground truth query and (2) the length (i.e., number of characters) of the ground truth query. For each group, we compute the average of the above metrics for all cases in the group.

We document the following information in the benchmark: (1) target schema and domain-specific explanations for attributes; (2) for each source dataset, its schema, domain-specific explanation of attributes, examples of instances, schema change hints for transforming the source table to the target format, and the ground truth query that transforms the source to the target. The benchmark dataset is open-sourced in a GitHub repository [1].

2) Other Benchmarks Used: We also used two other benchmarks that go beyond the transformation of smart building

data for different purposes. A commercial benchmark consists of 16 cases used by the Auto-Pipeline baseline. Since Auto-Pipeline code is not publicly available, we apply our proposed approach (without and with domain-specific knowledge) to the benchmark and compare the results.

Another benchmark consists of four COVID-19 data transformation cases, which we used to further validate how well our methodology can generalize to other data transformation scenarios. It includes all four transformation cases observed in the Github commit history of a widely used real-world COVID-19 data repository maintained by John Hopkins University [31]. The attributes in the target data are *(Province/State, Country/Region, Last Update, Confirmed, Deaths, Recovered)*, which represent the state-level COVID-

TABLE I
DESCRIPTIONS OF BENCHMARK GROUPS

ID	Group Description	Target Schema	Target Description	#Sources	Source Descriptions
1	Daily Hour-Level Load Profile Transformation	Date 1:00 2:00 ... 24:00	Date is of the format DOW MM/DD/YY, such as 'Fri 01/01/2016'	10	Sources include load profiles captured per minute, per 5-minute, per 10-minute and per hour with different schemas and column names. Some example source schemas are as follows: Ex1. (DateTime LoadValue), where DateTime is a timestamp such as '2/22/2018 0:30'. Ex2. (Segment Date 1:00 2:00 ... 24:00:00), where Date is in the format of DOW MM/DD/YY such as 'Wed 01/01/2003', and Segment is an attribute that should be discarded from the relation.
2		DT DOW HOURLY_0100 HOURLY_0200 ... HOURLY_2400 HOURLY_2500	DT is in the MM/DD/YY format. DOW has values from 1-7 corresponding to Mon-Sun. Hourly_2500 is used for leap second.	10	
3		Date Hour1 Hour2 ... Hour24	Date is of the format of MM/DD/YYYY	10	
4		Date 1:00AM 2:00AM ... 12:00PM	Date is of the format of MM/DD/YYYY	10	
5		Date Value1 Value2 ... Value24	Date is of the format of MM/DD/YYYY	10	
6		Date Hr1 Hr2 ... Hr24	Date is of the format of MM/DD/YYYY	10	
7	Monthly Hour-Level Load Profile Transformation	Month DayType HR1 HR2 ... HR24	Month has values such as January, Feburary, etc. DayType can be weekday or weekend.	10	
8		Hour January February ... December	Hour has values from 1 to 24. January records the average load in the corresponding hour in January. Other columns are similar.	10	
9	Seasonal Temperature Range Transformation	Season DayType Hour Temperature_Range Constant Coefficient Low_End High_End	Season has values such as SPRING, SUMMER, FALL, and WINTER. DayType can be either WEEKDAY or WEEKEND. Low_End is the lowest temperature. High_End is the highest temperature.	3	Source 1 and 2 are hourly temperature data grouped in four ranges and five ranges, respectively. Source 3 is seasonal temperature data grouped in three ranges.
10	Daily Hour-Level Load Transformation by Detailed Enduse	Datetime HVAC water_heating Refrigerator Clothes_washer Clothes_dryer Dishwasher Cooking_range Pool_spa_pumps Interior_lighting Exterior_lighting Lighting Plug Pool_spa_heater	The Datetime attribute has values in the format of YYYY-MM-DD HH:00:00.	3	The three sources are hourly datasets with 34, 151, and 32 attributes, respectively, mapped to 13 detailed end-uses. For example, the sum of electricity_pool_pump_kwh' and 'electricity_hot_tub_pump_kwh in source-1 is mapped to Pool_spa_pumps in the target.
11	Monthly Hour-Level Load Transformation by Detailed Enduse	Month Hour HVAC water_heating Refrigerator Clothes_washer Clothes_dryer Dishwasher Cooking_range Pool_spa_pumps Interior_lighting Exterior_lighting Lighting Plug Pool_spa_heater	Month is an integer number from 1 to 12. Hour is an integer number from 0 to 24.	4	Similar to Group 10, with one additional seasonal source dataset with an End Use Category attribute of which the values map to the 13 detailed end uses.
12	Seasonal Hour-Level Load Transformation by Detailed Enduse	Season Hour HVAC water_heating Refrigerator Clothes_washer Clothes_dryer Dishwasher Cooking_range Pool_spa_pumps Interior_lighting Exterior_lighting Lighting Plug Pool_spa_heater	Season has values such as Spring, Summer, Fall, and Winter. Hour" is an integer number from 0 to 24.	4	Similar to Group 11.
13	Daily Hour-Level Load Transformation by High-Level Enduse	Datetime HVAC Domestic_water_heating Major_appliances Lighting Miscellaneous_plug_loads Total	Similar to Group 10, except that this target has fewer (higher-level) end-uses.	3	Similar to Group 10.
14	Monthly Hour-Level Load Transformation by High-Level Enduse	Month Hour HVAC Domestic_water_heating Major_appliances Lighting Miscellaneous_plug_loads Total	Similar to Group 11, except that this target has fewer (higher-level) end-uses.	4	Similar to Group 11.
15	Seasonal Hour-Level Load Transformation by High-Level Enduse	Season Hour HVAC Domestic_water_heating Major_appliances Lighting Miscellaneous_plug_loads Total	Similar to Group 12, except that this target has fewer (higher-level) end-uses.	4	Similar to Group 12.

19 statistics. The source schemas of the first two cases involve county-level data with different numbers of columns, and the latter two cases involve state-level data with different column names and different numbers of columns.

Overall, we have tested 125 cases in three benchmarks, among which, 27 cases involve attribute merging, 89 cases involve attribute name changes, 32 cases involve pivoting, 5 cases involve attribute flattening, 50 cases involve group-by and aggregation, 8 cases involve join.

C. Evaluation Metrics

We report the following metrics in the experimental study:

• Execution Accuracy: This metric is defined as the ratio of the number of correctly transformed cases to the total number of transformation cases. For each case, if the LLM can return the correct transformation query that passes the experimental validation tests as described in Sec. III-C within 5 iterations, it is considered a correctly transformed case.

• Column Similarity: We compute the similarity score for each column in the transformed dataset and its corresponding column in the ground truth target dataset (defined in Sec. III-C). As detailed in Sec. III-C, we compute a similarity score for each column. We further define the *column similarity per case* as the average similarity scores of all the target attributes

End Uses	Sub End Uses	Source14_3
HVAC	Heating	CMPPWR
	Cooling	AHUPWR
	Furnace/AC fan	MSPLIT
	Boiler pumps	
	Kitchen range exhaust fan	
	Bath exhaust fan	
Domestic water heating	Domestic water heating	DWHPWR
Major appliances	Refrigerator	FRIDG1
	Clothes washer	NA
	Clothes dryer	DRY1CT DRY2CT
	Dishwasher	DSHWSR
	Cooking range	RNG1CT
	Pool/spa pumps	POOLPW
	Pool/spa heaters	NA
Lighting	Interior	NA
	Exterior	
Miscellaneous plug loads	Miscellaneous plug loads	WUPWHR
	Other refrigerators	FRIDG2
	Car	NA
Total		BLDPWR

No	Enduse Code	Description
1	BLDPWR	TOTAL BLDG WHS [emon ch1,2]
2	CMPPWR	AC COMPRESSOR WHS [emon ch3,4]
3	AHUPWR	AIR HANDLER WHS [emon ch5,6]
4	DWHPWR	WATER HEATER WHS [emon ch7,8]
5	DRY1CT	DRYER WHS (1-CT) [emon ch9]
6	RNG1CT	RANGE WHS (1-CT) [emon ch10]
7	DSHWSR	DISH WASHER WHS [emon ch11]
8	FRIDG1	Primary Fridge WHS [emon ch12]
9	FRIDG2	2nd Fridge WHS [emon ch13]
10	SPARE1	SPARE1 WHS (1-CT) [emon ch14]
11	SPARE2	SPARE2 WHS (1-CT) [xpod chA-1]
12	SPARE3	SPARE3 WHS (1-CT) [xpod chA-2]
13	POOLPW	POOL PUMP WHS (2-CTs) [xpod chA-3,4]
14	SPARE4	SPARE4 WHS (2-CTs) [xpod chA-5,6]
15	MSPLIT	Minisplit WHS (2-CTs) [xpod chA-7,8]
16	DRY2CT	Dryer WHS (2-CTs) [xpod chA-9,10]
17	OTHPWR	Calculated Unmeasured loads (Whr)
18	BLDPWC	Calculated Energy Use (Whr)
19	EXTRA1	Future use- WHS (2-CTs)[xpod chB-3,4]
20	EXTRA2	Future use- WHS (2-CTs)[xpod chB-5,6]
21	EMTEMP	eMonitor Temp (deg. F)
22	WUPCUM	Wattsup Cumulative Ent.Ctr (WHS)
23	WUPWHR	Wattsup Energy Ent.Ctr (WHS)
24	WUCUMW	Wattsup Cumulative Washer (WHS)
25	WUWASH	Wattsup Energy Washer (WHS)
26	LCTMP1	LaCrosse Device Temp (deg. F)
27	P6TEMP	PointSix Temp (deg. F)
28	P6HUMI	PointSix Humidity (%)
29	HBTEMP	HOBO LOGGER TEMP (deg. F)
30	HBRHUM	HOBO LOGGER RH (%)

Fig. 6. Example information from the smart building domain-specific database specifies the mapping from source attributes to the target attributes for the example in Fig. 5.

in the case, the *column similarity per group* as the average similarity scores of all cases in the group, and the *overall column similarity* as the average similarity scores of all the cases in all groups. The similarity score is set to zero for cases that fail to generate output data for similarity comparison.

• Number of Iterations to Success: For each case, we record *the number of iterations* used to achieve the correct response for the case. We record the *average number of iterations to success* for all successful cases that achieved a column similarity score of 1.0 within 5 iterations in each group, and in all groups. The latter is termed as the *overall number of iterations to success*.

D. Experimental Setups

We implemented the end-to-end workflow as illustrated in Fig. 2 in a Python script that uses the ChatGPT-3.5-turbor-16K model. We did not present results on ChatGPT-4, because the corresponding OpenAI API has a limit of 4K bytes for the total prompt-response size at this point, while this size is insufficient for a significant portion of real-world cases. For example, tables in Group 10 to Group 15 have up to 152 attributes, leading to a large prompt size. We set the temperature to zero to avoid randomness for several reasons. First, a primary goal of this work is to evaluate the effectiveness of LLMs on data transformation tasks by using different types of initial prompts and the effectiveness of iterative prompt optimization. Random responses require additional methods (e.g., majority voting) for self-consistency, which will complicate the comparison. Second, setting the temperature to zero will achieve better quality results in most cases, according to a recent OpenAI article [32]. All SQL codes are run on the PostgreSQL database version 15.0 for validation. All descriptions of the source and target attributes

are obtained from a domain-specific database[3].

E. Smart Building Data Transformation Results

1) Overall Results.: The zero-shot learning results for the smart building data transformation benchmark are illustrated in Tab. II. Using Prompt-3, our proposed `SQLMorpher` methodology achieved an execution accuracy of **96**%, which is significantly higher than Prompt-1 and Prompt-2, which achieved an execution accuracy of 28% and 36%, respectively. It demonstrated the importance of supplying domain-specific knowledge, particularly schema change hints, as part of the prompt to the LLM. The observation justifies the integration of the LLM with the domain-specific knowledge base and the schema mapping tools for data transformation pipelines.

2) Effectiveness of One-shot Learning.: For the four cases that failed with Prompt-3, we applied Prompt-4, Prompt-5, and Prompt-6 to check whether providing one demonstration example that involves a similar prompt and a correct response can improve the LLM response. The results are illustrated in Tab. III, which showed that using prompts that combine domain-specific knowledge and demonstration is capable of solving all four complicated cases that failed with Prompt-3.

3) Effectiveness of the Iterative Optimization Process.: Compared to Prompt-1 and Prompt-2, we have found that Prompt-3 can gain significantly more from iterative prompt optimization. When using Prompt-1, five cases in three groups, Group-1, Group-4, and Group-7, benefit from iterative prompt optimization, the average number of iterations being 1.2, 1.3, and 2.0, respectively, as illustrated in Tab. II. Other groups either have all cases passed in one iteration or have all cases failed. When using Prompt-2, four cases in three groups, Group-2, Group-3, and Group-7, require more than one iteration to succeed, the average number of iterations being 1.4, 1.2, and 1.2, respectively. When using Prompt-3, 10 cases in six groups, require more than one iteration to succeed. It means that 9.5% of total cases can benefit from iterative prompt optimization when using Prompt-3.

F. Results on Benchmarks Beyond Smart Building.

First, we tested our approach on the COVID-19 benchmark. The results are illustrated in Tab. IV, which showed that our proposed methodology resolves all four cases simply using the basic prompt (Prompt-1).

Second, we also compared our proposed approach with the Auto-Pipeline approach, using its commercial benchmark [4]. The results are illustrated in Tab. V. It showed that our proposed methodology achieved perfect execution accuracy on all 16 transformation problems in their benchmark only using the basic prompt, without additional domain-specific knowledge. The execution accuracy achieved by Auto-Pipeline on this benchmark is below 70% [4]. The comparison implies that our approach has great potential to outperform state-of-the-art automatic data transformation tools.

[3]The domain-specific database is maintained by co-author Liang Zhang. Some example information in the database is illustrated in Fig. 4 and Fig. 6.

TABLE II

COMPARISON OF EXECUTION ACCURACY USING DIFFERENT PROMPT TEMPLATES WITH ZERO-SHOT LEARNING (GRP STANDS FOR GROUP)

	Grp-1	Grp-2	Grp-3	Grp-4	Grp-5	Grp-6	Grp-7	Grp-8	Grp-9	Grp-10	Grp-11	Grp-12	Grp-13	Grp-14	Grp-15
#keywords avg.	15.6	18.7	19.6	19.6	16.4	16.3	24.2	26.7	23.7	13.7	20.3	28.0	5.0	25.3	31.5
length avg.	1802	2826	1957	2023	1731	1713	1548	3239	1034	1712	2085	2365	1412	1732	1918
Prompt 1. Overall execution accuracy: 29/105 (28%); Overall column similarity scores: 0.4; Overall iteration to success: 1.3															
exec auc	6/10	2/10	4/10	6/10	3/10	2/10	6/10	0/10	0/3	0/3	0/4	0/4	0/3	0/4	0/4
sim score avg.	0.7	0.5	0.6	0.6	0.3	0.3	0.8	0.0	0.0	0.6	0.3	0.0	0.3	0.0	0.0
iter-to-succ avg.	1.2	1.0	1.0	1.3	1.0	1.0	2.0	-	-	-	-	-	-	-	-
Prompt 2. Overall execution accuracy: 38/105 (36%); Overall column similarity score: 0.5; Avg iteration to success: 1.1															
exec auc	6/10	6/10	7/10	6/10	3/10	2/10	8/10	0/10	0/3	0/3	0/4	0/4	0/3	0/4	0/4
sim score avg.	0.7	0.6	0.7	0.6	0.4	0.3	0.9	0.0	0.4	0.8	0.7	0.2	0.3	0.1	0.0
iter-to-succ avg.	1.0	1.4	1.2	1.0	1.0	1.0	1.2	-	-	-	-	-	-	-	-
Prompt 3. Overall execution accuracy: 101/105 (96%); Overall column similarity score: 0.96; Avg iteration to success: 1.2															
exec auc	10/10	10/10	10/10	10/10	10/10	10/10	10/10	9/10	3/3	3/3	4/4	3/4	3/3	2/4	4/4
sim score avg.	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9	1.0	1.0	1.0	0.8	1.0	0.5	1.0
iter-to-succ avg.	1.0	1.0	1.1	1.0	1.7	1.6	1.0	1.0	1.0	1.3	1.5	1.0	1.3	1.0	1.0

TABLE III

AVERAGE COLUMN SIMILARITY SCORE WITH ONE-SHOT LEARNING

Cases Failed with Prompt-3	Prompt-1 +Demo	Prompt-2 +Demo	Prompt-3 +Demo
Case 78 (in Group 8)	1.00	1.00	1.00
Case 92 (in Group 12)	0.00	0.27	1.00
Case 100 (in Group 14)	0.38	0.50	1.00
Case 101 (in Group 14)	0.50	0.50	1.00

TABLE IV

PROMPT COMPARISON FOR COVID-19 BENCHMARK
#KEYWORDS AVG.: **5**, LENGTH AVG.: **277**

	Prompt-1	Prompt-2	Prompt-3
exec auc	4/4	4/4	4/4
sim score avg.	1.0	1.0	1.0
iter-to-succ avg.	1.0	1.0	1.0

TABLE V

COMPARISON TO AUTO-PIPELINE ON THEIR COMMERCIAL BENCHMARK
#KEYWORDS AVG.: **8**, LENGTH AVG.: **566**

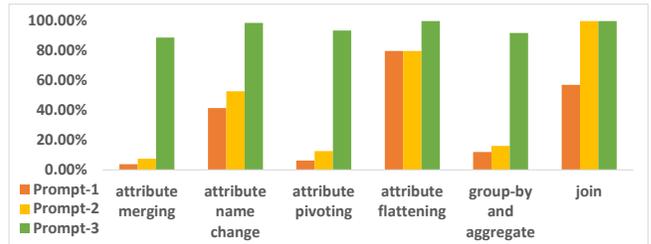|	Prompt-1	Prompt-2	Prompt-3	Auto-Pipeline [4]
exec auc	13/16	**16/16**	**16/16**	11/16 [4]
sim score avg.	0.83	1.00	1.00	-
iter-to-succ avg.	1.00	1.00	1.00	-

Fig. 7. The overall execution accuracy of cases in each schema change category. (We considered all 125 cases in three benchmarks; each case may involve multiple types of schema changes.)

G. Summary of Key Findings

• Large language models are promising to automatically resolve complicated smart building data transformation cases if domain-specific knowledge is available and easily retrievable. We achieved 96% accuracy on our proposed benchmark, which consists of 105 real-world smart building cases.

• Our `SQLMorpher` methodology is promising in generalizing to other data transformation cases and outperforming state-of-the-art automatic data transformation tools that do not rely on LLMs. In particular, our methodology defines clean interfaces for integrating domain-specific knowledge into the data transformation process through the prompt generation process. This is a missing feature in state-of-the-art data transformation tools. The evaluation results on the commercial benchmark used by Auto-Pipeline showed that our approach, even without using domain-specific knowledge, could achieve significantly better execution accuracy than Auto-Pipeline (81% vs. 69%). One observation is that while the LLM generates SQL code, Auto-Pipeline attempts to learn a pipeline of data transformation operators. The latter has a more limited search space,

which may affect execution accuracy.

• Compared to other domain-specific knowledge, a high-level schema change hint, such as column mapping relationships or instructions as simple as "use aggregation", is critical to the success of our proposed methodology.

• We further classify each of 125 cases from all three benchmarks into one or more schema change types. We then count the execution accuracy for each type of schema change, as illustrated in Fig. 7. We observe that while Prompt-3 with schema change hints can handle all schema change types well, Prompt-1 and Prompt-2 without schema change hints achieved relatively better accuracy (40% to 100%) for attribute name change, attribute flattening, and join than other types of changes, such as attribute merging, attribute pivoting, and group-by/aggregation. This further verified the importance of incorporating high-level schema change hints such as "use aggregation" and "use pivoting".

- Zero-shot learning is effective in resolving most data transformation problems investigated in this work. Few-shot learning can resolve the difficult cases that fail with zero-shot.

- The iterative optimization framework that simply enhances the prompt using ChatGPT reported errors or SQL execution errors for each iteration can benefit 9.5% of cases when using Prompt-3 and 5% of cases when using Prompt-1 and 2.

- The examples in our proposed building energy data transformation benchmark are significantly more complicated than existing benchmarks in terms of the number of distinct keywords and the length of the transformation query. They are used in the real world, but are missing in existing data transformation benchmarks [4], [5].

V. CONCLUSION AND FUTURE WORKS

In this work, we pioneered the experimental and feasibility study about applying LLMs to data transformation problems. We proposed a novel `SQLMorpher` approach using LLMs to generate SQL modification queries for data transformation. `SQLMorpher` is designed to flexibly incorporate domain knowledge and optimize prompts iteratively. We provided a unique benchmark for building energy data transformation, including 105 real-world cases collected from 21 energy companies in the United States. The results are promising, achieving up to **96**% accuracy on the benchmark. In addition, we have found our system can generalize to scenarios beyond building energy data. The commercial benchmark results demonstrate that our approach is able to outperform existing automatic data transformation techniques significantly. In summary, `SQLMorpher` is promising to enable the automatic integration of diverse data sources for building energy management and may benefit other domains.

In the future, we will design automated quality control for `SQLMorpher` further to reduce human validation involvement in the production environment. We will integrate `SQLMorpher` with downstream tasks, such as building energy management simulation, and investigate more complicated data transformation cases in the building sector, such as energy measure/unit transformation, timezone transformation, etc. At the same time, our work can be used to enhance existing works. For example, by applying our work, we can extend state-of-the-art Text2SQL tools to support the insert statements and data transformation scenarios. It can also automate PowerBI, and other interactive transformation tools by generating detailed data transformation instructions.

ACKNOWLEDGMENT

REFERENCES

[1] Energy Information Administration, "U.S. energy consumption by source and sector, 2022." https://www.eia.gov/totalenergy/data/monthly/pdf/flow/total_energy_2022.pdf.

[2] G. Pinto, Z. Wang, A. Roy, T. Hong, and A. Capozzoli, "Transfer learning for smart buildings: A critical review of algorithms, applications, and future perspectives," *Advances in Applied Energy*, vol. 5, p. 100084, 2022.

[3] Z. Jin, Y. He, and S. Chauduri, "Auto-transform: learning-to-transform by patterns," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 2368–2381, 2020.

[4] J. Yang, Y. He, and S. Chaudhuri, "Auto-pipeline: synthesizing complex data pipelines by-target using reinforcement learning and search," *Proceedings of the VLDB Endowment*, vol. 14, no. 11, pp. 2563–2575, 2021.

[5] P. Li, Y. He, C. Yan, Y. Wang, and S. Chauduri, "Auto-tables: Synthesizing multi-step transformations to relationalize tables without using examples," *arXiv preprint arXiv:2307.14565*, 2023.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[7] Z. Wang, L. Zhou, A. Das, V. Dave, Z. Jin, and J. Zou, "Survive the schema changes: integration of unmanaged data using deep learning," *arXiv preprint arXiv:2010.07586*, 2020.

[8] I. Trummer, "Codexdb: Synthesizing code for query processing from natural language instructions using gpt-3 codex," *Proceedings of the VLDB Endowment*, vol. 15, no. 11, pp. 2921–2928, 2022.

[9] Z. Gu, J. Fan, N. Tang, S. Zhang, Y. Zhang, Z. Chen, L. Cao, G. Li, S. Madden, and X. Du, "Interleaving pre-trained language models and large language models for zero-shot nl2sql generation," *arXiv preprint arXiv:2306.08891*, 2023.

[10] Z. Gu, J. Fan, N. Tang, L. Cao, B. Jia, S. Madden, and X. Du, "Few-shot text-to-sql translation using structure and content prompt learning," *Proceedings of the ACM on Management of Data*, vol. 1, no. 2, pp. 1–28, 2023.

[11] O. Popescu, I. Manotas, N. P. A. Vo, H. Yeo, E. Khorashani, and V. Sheinin, "Addressing limitations of encoder-decoder based approach to text-to-sql," in *Proceedings of the 29th International Conference on Computational Linguistics*, pp. 1593–1603, 2022.

[12] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[13] G. Fan, J. Wang, Y. Li, D. Zhang, and R. J. Miller, "Semantics-aware dataset discovery from data lakes with contextualized column-based representation learning," *Proceedings of the VLDB Endowment*, vol. 16, no. 7, pp. 1726–1739, 2023.

[14] Y. Dong, K. Takeoka, C. Xiao, and M. Oyamada, "Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 456–467, IEEE, 2021.

[15] L. Wang, S. Zhang, J. Shi, L. Jiao, O. Hassanzadeh, J. Zou, and C. Wangz, "Schema management for document stores," *Proceedings of the VLDB Endowment*, vol. 8, no. 9, pp. 922–933, 2015.

[16] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, *et al.*, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task," *arXiv preprint arXiv:1809.08887*, 2018.

[17] T. Shi, C. Zhao, J. Boyd-Graber, H. Daumé III, and L. Lee, "On the potential of lexico-logical alignments for semantic parsing to sql queries," *arXiv preprint arXiv:2010.11246*, 2020.

[18] X. Yu, T. Chen, Z. Yu, H. Li, Y. Yang, X. Jiang, and A. Jiang, "Dataset and enhanced model for eligibility criteria-to-sql semantic parsing," in *12th International Conference on Language Resources and Evaluation (LREC)*, 2020.

[19] C.-H. Lee, O. Polozov, and M. Richardson, "Kaggledbqa: Realistic evaluation of text-to-sql parsers," *arXiv preprint arXiv:2106.11455*, 2021.

[20] Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stone-braker, "Dataxformer: A robust transformation discovery system," in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pp. 1134–1145, IEEE, 2016.

[21] S. Gulwani, "Automating string processing in spreadsheets using input-output examples," *ACM Sigplan Notices*, vol. 46, no. 1, pp. 317–330, 2011.

[22] Y. He, X. Chu, K. Ganjam, Y. Zheng, V. Narasayya, and S. Chaudhuri, "Transform-data-by-example (tde) an extensible search engine for data transformations," *Proceedings of the VLDB Endowment*, vol. 11, no. 10, pp. 1165–1177, 2018.

[23] J. Heer, J. M. Hellerstein, and S. Kandel, "Predictive interaction for data transformation.," in *CIDR*, Citeseer, 2015.

[24] Z. Jin, M. Cafarella, H. Jagadish, S. Kandel, M. Minar, and J. M. Hellerstein, "Clx: Towards verifiable pbe data transformation," *arXiv preprint arXiv:1803.00701*, 2018.

[25] R. Singh, "Blinkfill: Semi-supervised programming by example for syntactic string transformations," *Proceedings of the VLDB Endowment*, vol. 9, no. 10, pp. 816–827, 2016.

[26] Z. Jin, M. R. Anderson, M. Cafarella, and H. Jagadish, "Foofah: Transforming data by example," in *Proceedings of the 2017 ACM International Conference on Management of Data*, pp. 683–698, 2017.

[27] E. Zhu, Y. He, and S. Chaudhuri, "Auto-join: Joining tables by lever-aging transformations," *Proceedings of the VLDB Endowment*, vol. 10, no. 10, pp. 1034–1045, 2017.

[28] C. Zuo, S. Assadi, and D. Deng, "Spine: Scaling up programming-by-negative-example for string filtering and transformation," in *Proceedings of the 2022 International Conference on Management of Data*, pp. 521–530, 2022.

[29] L. T. Becker and E. M. Gould, "Microsoft power bi: Extending excel to manipulate, analyze, and visualize diverse data," *Serials Review*, vol. 45, no. 3, pp. 184–188, 2019.

[30] Trifacta, "Trifacta wrangler," 2020.

[31] Johns Hopkins Coronavirus Resource Center, "Covid-19 data repository by the center for systems science and engineering (csse) at johns hopkins university.." https://github.com/CSSEGISandData/COVID-19.

[32] "Codex models and azure openai service." https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/work-with-code.