ELT-Bench: An End-to-End Benchmark for Evaluating AI Agents on ELT Pipelines

Tengjun Jin
University of Illinois (UIUC)
Urbana, USA
tengjun2@illinois.edu

Yuxuan Zhu
University of Illinois (UIUC)
Urbana, USA
yxx404@illinois.edu

Daniel Kang
University of Illinois (UIUC)
Urbana, USA
ddkang@illinois.edu

ABSTRACT

Practitioners are increasingly turning to Extract-Load-Transform (ELT) pipelines with the widespread adoption of cloud data warehouses. However, designing these pipelines often involves significant manual work to ensure correctness. Recent advances in AI-based methods, which have shown strong capabilities in data tasks, such as text-to-SQL, present an opportunity to alleviate manual efforts in developing ELT pipelines. Unfortunately, current benchmarks in data engineering only evaluate isolated tasks, such as using data tools and writing data transformation queries, leaving a significant gap in evaluating AI agents for generating end-to-end ELT pipelines.

To fill this gap, we introduce ELT-Bench, an end-to-end benchmark designed to assess the capabilities of AI agents to build ELT pipelines. ELT-Bench consists of 100 pipelines, including 835 source tables and 203 data models across various domains. By simulating realistic scenarios involving the integration of diverse data sources and the use of popular data tools, ELT-Bench evaluates AI agents' abilities in handling complex data engineering workflows. AI agents must interact with databases and data tools, write code and SQL queries, and orchestrate every pipeline stage. We evaluate four representative code agents with six popular Large Language Models (LLMs) on ELT-Bench. The highest-performing agent, OpenHands CodeActAgent Claude-3.5-Sonnet, correctly generates only 11.3% of data models, with an average cost of $1.41 and 72.2 steps per pipeline. Our results demonstrate the challenges of ELT-Bench and highlight the need for a more advanced AI agent to reduce manual effort in ELT workflows.

PVLDB Reference Format:
Tengjun Jin, Yuxuan Zhu, and Daniel Kang. ELT-Bench: An End-to-End Benchmark for Evaluating AI Agents on ELT Pipelines. PVLDB, 19(2): 84 - 98, 2025.
doi:10.14778/3773749.3773750

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/uiuc-kang-lab/ELT-Bench/tree/eltbench.

Proceedings of the VLDB Endowment, Vol. 19, No. 2 ISSN 2150-8097.
doi:10.14778/3773749.3773750

1 INTRODUCTION

Data engineers are increasingly leveraging Extract-Load-Transform (ELT) pipelines to integrate data and efficiently transform it into the required format as scalable cloud data warehouses become more accessible and storage prices continue to fall [22, 43, 56, 60, 61]. For example, the TPC-DI benchmark requires the creation of a decision support system for a retail brokerage firm by transforming data from various sources, including a trading system, internal Human Resources (HR), and Customer Relationship Management (CRM) systems [48]. These data sources vary in formats, data types, attributes, and inter-table relationships [48]. To build such a decision support system, data engineers design ELT pipelines: first, extracting and loading data into the data warehouse, followed by writing transformation queries to process the data for analysis.

Compared to traditional Extract-Transform-Load (ETL) pipelines, ELT pipelines ingest data directly into data warehouses, enabling real-time Business Intelligence (BI) analysis [66]. Furthermore, with cloud infrastructure, ELT enhances scalability for processing large volumes of data [60] and offers greater flexibility in incorporating additional data transformations [49]. These benefits make ELT pipelines an increasingly preferred choice for processing data across various scenarios [22, 43, 56, 60, 61].

Developing ELT pipelines is an essential task for data engineers [22, 43, 56, 60, 61], but the process requires significant manual work. Prior studies estimate that data engineers spend over 60% of their time on data warehousing projects building data pipelines [11, 26, 34, 54, 72]. First, these pipelines must extract and integrate data from disparate sources with varying formats and standards. Second, data engineers or analysts need a deep understanding of the source data schema to write transformation queries.

Can AI agents effectively reduce the manual effort involved in constructing ELT pipelines? Recent advancements in Large Language Models (LLMs) have demonstrated strong capabilities in the text-to-SQL task, a crucial component of ELT pipelines. Notably, state-of-the-art (SOTA) techniques based on LLMs have achieved execution accuracy rates of 77.1% and 91.2% on the Bird [40] and Spider 1.0 [81] benchmarks, respectively. Researchers have recently developed AI agents to tackle more complex real-world tasks that demand reasoning, tool usage, planning, and memorization [38, 58, 68, 71, 76, 79]. To evaluate the capability of emerging AI agents, researchers have proposed numerous benchmarks in the data domain [9, 28, 30, 37, 38]. However, there is no end-to-end benchmark designed with end-to-end ELT pipelines.

Building an end-to-end ELT benchmark is challenging because it requires sophisticated setup and configuration, time-consuming ground truth labeling, and thorough workflow verification to ensure reproducibility and correctness. First, annotators must set up

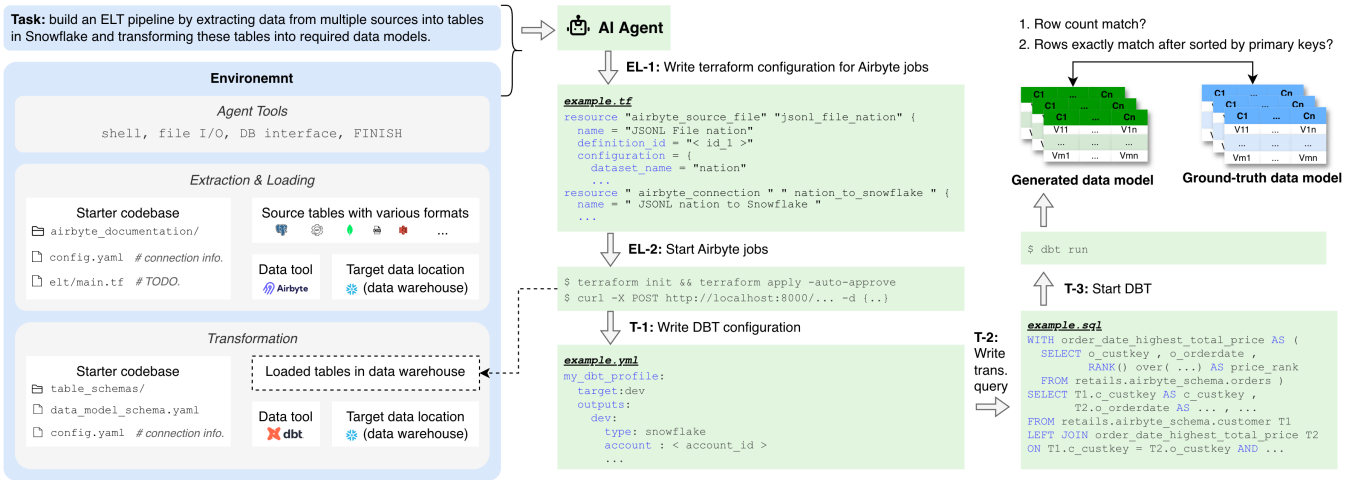


Figure 1: ELT-Bench is the first end-to-end benchmark designed to evaluate the ability of AI agents to build ELT pipelines. The agent is required to build ELT pipelines by setting up `Airbyte` to ingest data from diverse sources, and then writing configuration files and SQL transformation queries in `DBT` to produce analysis-ready data models.

various data management systems and platforms to store source data and provide data tools capable of handling diverse data sources. Second, annotators must prepare all necessary input files within the project base. Third, annotators must label ground truth by developing configuration files and writing complex transformation queries involving various relational operations (e.g., casting, type conversion, joins, aggregation, and ranking). Finally, annotators manually execute and verify each ELT pipeline to validate the correctness of both the configured environment and generated annotations.

We address these challenges by introducing ELT-Bench, a new benchmark of 100 ELT pipelines associated with 835 source tables and 203 data models across various domains. For a single ELT pipeline, we spend approximately 3 to 5 hours of manual effort setting up the environment, annotating input files and the ground truth, and building the entire pipeline for verification. Notably, 60% of the pipelines require extracting and integrating data from five distinct categories of sources (APIs, cloud services, relational databases, NoSQL databases, and flat files). In addition, the ground truth for each pipeline, on average, involves 187 lines of code per configuration file and 200 SQL tokens (tokenized by whitespace [38]) per data model.

ELT-Bench is the first benchmark that covers the entire workflow for building ELT pipelines. As shown in Figure 1, ELT-Bench requires agents to construct an end-to-end ELT pipeline from scratch, encompassing two primary stages: (1) data extraction & loading stage and (2) data transformation stage. In the first stage, agents use provided connection details (Figure 2a) to write and execute `Airbyte Terraform` code, extracting data from diverse sources and loading it into the data warehouse. In the second stage, agents configure `DBT` (including its profile and SQL transformation queries) and execute it to produce target data models. Once the entire ELT pipeline is complete, analysts can directly access both ingested source tables and target data models in the warehouse for further analysis. This setup challenges AI agents to automate the end-to-end data engineering workflow required for building ELT pipelines.

We evaluate four code agents, Spider-Agent [38], SWE-Agent [76], OpenHands CodeActAgent [68], and Augment Agent [12], with six LLMs on ELT-Bench. The top-performing agent, OpenHands CodeActAgent Claude-3.5-Sonnet, achieves a success rate of 73% in the data extraction & loading stage but only a success rate of 11.3% in the data transformation stage. On average, OpenHands CodeActAgent Claude-3.5-Sonnet consumes $1.41 and requires 72.2 execution steps per task. Current agents' poor performance and high costs highlight the need for further advancements in AI agents to reduce manual effort in developing ELT pipelines.

2 ELT-BENCH

In this section, we first provide an overview of ELT-Bench, detailing the data tools, the overall task, the agent's input and output, and the execution environment. We then present summary statistics of ELT-Bench.

2.1 ELT-Bench Overview

Data tools. We use `Airbyte` [2] for data extraction and loading, a leading open-source data integration tool for ELT pipelines, which has also been used in prior work [9]. We use the `Airbyte Terraform` provider to enable code-based configuration for the agent. To generate target data models, we use `DBT` [14], a widely adopted data transformation tool [9, 38].

Task description. Given a codebase and environment, the agent is tasked with editing the codebase to implement `Airbyte` configuration, `DBT` configuration, and data transformation queries, as well as running the codebase to construct an end-to-end ELT pipeline. The task comprises two main stages:

(1) *Data extraction & loading:* The agent must first write `Airbyte Terraform` code and then trigger extraction & loading jobs to ingest data from various sources into the data warehouse.

```yaml
Airbyte:
  config:
    files_definition_id: <id_1>
    workspace_id: <id_2>
flat_files:
- format: jsonl
  path: "https://..."
  sync_mode: full_refresh_append
  table: nation
snowflake:
  config:
    account: <account_id>
    database: retails
    password: <snowflake_password>
    username: AIRBYTE_USER ...
```

(a) A partial configuration of `Airbyte` for the `retails` database, as defined in the provided `config.yaml` file.

```yaml
models:
- name: customers
  description: Each record represents a customer.
  columns:
  - name: c_custkey
    description: Unique identifier for the customer.
  - name: order_date_highest_total_price
    description: The order date with the highest total
         price the customer has made, with ties broken by
          the ascending order of the order date. ...
```

(b) A partial description of the `customers` data model for the `retails` database, as defined in the provided data_model.yaml file.

Figure 2: An example of provided input files of the `retails` database in ELT-Bench.

(2) *Data transformation:* The agent is then required to configure DBT and write transformation queries. Subsequently, the agent needs to execute DBT to produce analysis-ready data models in the data warehouse.

Agent inputs. We now describe the details of the pre-established project base, which consists of the following files:

(1) *Source tables* refers to the original tables in diverse formats, such as relational databases, APIs, and flat files.
(2) `config.yaml` contains the necessary connection information for source tables, data warehouses, `Airbyte`, and DBT. For example, extracting data from `PostgreSQL` requires specifying the host, port, user, password, schema, database, and tables.
(3) `data_model.yaml` defines the data models the ELT pipeline generates. Each data model includes a description, column names, and explanations for each column.
(4) `elt/main.tf` contains the code to initialize `Terraform` provided in `Airbyte`.
(5) `documentation` includes correct version of documentation from `Airbyte`, providing guidance on writing configuration code and triggering sync jobs.
(6) *schemas* contains the column names and descriptions of source tables.

Agent outputs. We now detail the agent's output, which consists of three components:

(1) *Files in codebase:* Newly created `Airbyte` `Terraform` code, DBT profile files, and SQL transformation queries.
(2) *Loaded tables:* Source data ingested into the data warehouse and stored in a unified format.
(3) *Data models:* Analyst-required tables produced by applying SQL transformations (e.g., join, filter, aggregation) to the loaded tables.

Enviroment description. ELT-Bench also includes a complex environment, as it requires agents to interact with a variety of data storage platforms and data tools. We describe them in detail below:

(1) *Data sources*: We select five commonly used types of data storage platforms for storing source tables, as listed in Table 3. We deploy four of these platforms—`PostgreSQL`, `MongoDB`, `REST API`, and `Amazon S3` (simulated with `LocalStack` [41])—using Docker containers, while providing links for flat files.
(2) *Data warehouse*: We use `Snowflake` [13] as our data warehouse to store extracted data and execute transformation queries that generate target data models, as it is a widely studied and popular cloud data warehousing solution [38, 64].
(3) *Packages and functions*: We provide a Docker file with all the required packages. Additionally, since data extraction and loading jobs typically take several minutes, we provide a script that monitors the status of all synchronization jobs and waits for their completion. This prevents redundant `Airbyte` API and LLM calls, reducing execution steps and costs.

2.2 Benchmark Statistics

ELT-Bench contains 100 ELT pipelines associated with 835 source tables and 203 data models. As shown in Table 1, compared to existing agent benchmarks for data engineering, ELT-Bench is the first end-to-end benchmark that covers the full data engineering workflow, reflecting real-world practice. In contrast, TPC-DI [48] is limited to a single database and its metrics are designed to assess data integration system throughput rather than AI agents' ability. Spider 2-V [9] focuses on evaluating an agent's ability to use high-level data tools individually, such as using `Airbyte` to extract and load data and using DBT with a given SQL query to transform data. It does not include writing low-level SQL queries for data transformation or creating a complete pipeline using multiple tools. Spider 2.0 [38] includes data transformation in only 10.8% of tasks and does not cover extraction or loading. We exclude recent benchmarks built on Bird [40] (i.e., TAG-Bench [6] and TQA-Bench [52]) from Table 1 because they focus on data-analytic tasks (i.e., question answering) and do not include data extraction and loading or data transformation. We highlight the characteristics of ELT-Bench in the following paragraphs.

Diverse Data sources. As shown in Tables 2 and 3, our benchmark features diverse data sources. In total, 60 pipelines require extracting data from all five types of data sources, while 24 pipelines involve extracting more than 10 tables. Furthermore, 30 pipelines require writing more than 200 lines of code in `Terraform` files to extract data from these sources and load them into the data warehouse.

Complex Data Transformation. ELT-Bench evaluates the agent's ability to write SQL queries based on natural language to generate

Table 1: Comparison of ELT-Bench with existing benchmarks in the data engineering field. ELT-Bench is the first benchmark to evaluate AI agents' ability to provide end-to-end coverage of the entire data integration process, from ingesting tables in diverse formats to analysis-ready data models in the data warehouse.

Benchmark	# Tasks	Data Extraction & Loading	Data Transformation	End-to-End
TPC-DI [48]	1	✓ (12)	✓ (18)	✓ (1)
Spider2-V [9]	494	✓ (48)	×	×
Spider 2.0 [38]	632	×	✓ (120)	×
ELT-Bench	100	✓ (835)	✓ (203)	✓ (100)

Table 2: Statistics of ELT-Bench, illustrating the distribution of data sources, source tables, lines of `Terraform` code, data models, and SQL tokens per data model. As shown, ELT-Bench consists of ELT pipelines that involve multiple data sources, extensive code, and complex data transformations.

Statistics	# Tasks
# Categories of Data Sources	100
2 data sources	7
3 data sources	15
4 data sources	18
5 data sources	60
# Source Tables	100
< 5 tables	36
5 - 10 tables	40
> 10 tables	24
# Lines of Airbyte Terraform Code	100
< 100 lines	7
100 – 200 lines	63
> 200 lines	30
# Target Data Models	100
1 data model	50
2 data models	22
≥ 3 data models	28
# SQL Tokens per Data Model (Tokenized by whitespace [38])	100
< 100 tokens	8
100–200 tokens	19
> 200 tokens	73

target data models. It is common practice in ELT workflows to generate multiple data models within a single task. For example, TPC-DI requires generating 18 data models [48]. In ELT-Bench, for example, 28 pipelines require the generation of at least three data models each. Following the approach in Spider 2.0 [38], we tokenize the SQL queries using whitespace and then count the resulting tokens to measure complexity. Because pipelines from `Fivetran` include both a staging and an intermediate layer, we calculate the average number of tokens per data model for each pipeline. As shown in Table 2, 73 pipelines demand over 200 tokens per data model, illustrating the complexity of these SQL queries.

Table 3: Overview of common data source categories, representative sources, and their real-world applications.

Data Source Category	Representative Sources	Applications in Practice
APIs	`REST API`	Web services, third-party platforms, real-time applications.
Cloud Services	`Amazon S3`	Big data platforms, modern applications.
Relational Databases	`PostgreSQL`	Enterprise systems, transactional systems.
NoSQL Databases	`MongoDB`	Modern web applications, real-time data systems.
Flat Files	CSV, JSONL, Parquet	Third-party data providers, backups.

3 ELT-BENCH CONSTRUCTION

3.1 Data Collection

We collect 78 databases from a widely used text-to-SQL benchmark, Bird [40], and 22 databases from the GitHub repository[1] of an enterprise software, `Fivetran` [21]. Each database corresponds to a pipeline in ELT-Bench.

- Bird is a text-to-SQL benchmark with large-scale databases spanning 37 domains. We use all databases from which five distinct features can be extracted as columns within a data model, resulting in the inclusion of 78 out of 80 databases in ELT-Bench. Previous study indicates that databases in Bird can contain noise levels as high as 49% [73]. To ensure quality, we manually verify the SQL queries for the five features used in our benchmark, correcting any identified errors directly in our gold SQL transformation queries.
- `Fivetran` is a data movement platform that develops `DBT` packages to facilitate the analysis of data from popular sources, such as Microsoft Advertising, Instagram Business, and YouTube Analytics. We sampled 22 out of 103 packages from the `Fivetran` GitHub repository.

3.2 Annotation Pipeline

We now describe the construction of the ELT-Bench environment and codebase, as well as the annotation and verification of the

[1]Fivetran releases their data models on GitHub: https://github.com/fivetran.

ground truth. We defer the detailed steps to the supplementary material [33].

Environment setup. The main step in the environment setup is constructing diverse data sources for each ELT task. The original Bird data is stored in `SQLite`, while `Fivetran` data is in CSV format. To simulate real-world diversity, we convert these data sources into different formats according to their characteristics and the categories shown in Table 3. To facilitate the use of ELT-Bench, we use Docker to deploy `PostgreSQL`, `MongoDB`, `REST API`, and `Amazon S3`. Since a Docker image containing a full database dump would be large, we instead provide an image with scripts to create the necessary databases and load the data.

Codebase construction. After setting up the environment, we construct the ELT-Bench codebase. We first specify the connection details for each data source, data warehouse, and data tool in `config.yaml` (see Figure 2a for an example).

We define 203 ELT data models: 80 derived from `Fivetran` databases and 123 from Bird databases. For `Fivetran`, which provides predefined data models, we prune each data model by dropping utility-generated columns and columns that are always null, except when such columns are required by downstream data models. For Bird, a text-to-SQL benchmark with up to several hundred annotated analytical questions per database, we define the target data models for the ELT pipelines following the data models from TPC-DI [48] and `Fivetran`. For each database from Bird, we group annotated analytical questions by the dimension table they target. Then for each dimension table whose associated questions yield at least five distinct features, we randomly sample five questions involving complex SQL and convert them into the implied features as columns in the corresponding data model. For example, consider the question: "Which film directed by Abbas Kiarostami has the highest average score?" This corresponds to the feature `highest_average_score_film` in the `Directors` data model, representing the film with the highest average score for each director. We include a textual description for each column to help agents better understand the data model and reduce ambiguity.

Ground truth annotation and verification. Annotators are required to consult the official `Airbyte Terraform` documentation to learn the configuration process for `Airbyte`. They then write the necessary `Airbyte Terraform` code and SQL queries based on the codebase. Although Bird provides ground truth SQL queries, annotators must verify the correctness of the queries we use, fix any errors, and modify them to conform to the defined data models.

After annotation, we validate the ground truth by executing the codebase to construct ELT pipelines and inspecting the output data models. We run, on average, 10 test queries for each data model to ensure correctness. If the output data models do not match expectations, we revise the ground truth and re-run the codebase until the outputs meet the desired criteria.

4 EXPERIMENTS

We evaluate four representative code agent frameworks, Spider-Agent [38], SWE-Agent [76], OpenHands CodeAct Agent [69], and Augment Agent [12] using six LLMs on ELT-Bench. In this section, we first introduce the evaluation metrics of ELT-Bench, followed by a detailed explanation of the experimental settings for both agents. Finally, we present the evaluation results.

4.1 Evaluation Metrics

We use the widely adopted metric in agentic benchmarks [9, 29, 32, 38, 84], success rate, to assess the performance of agents on ELT-Bench. To provide a more comprehensive evaluation, we measure the success rate for both the data extraction & loading stage and the data transformation stage. Specifically, we introduce the **Success Rate for Data Extraction & Loading (SRDEL)** to measure the proportion of ELT pipelines that successfully extract and load data in the first stage and the **Success Rate for Data Transformation (SRDT)** to measure the proportion of data models successfully built in the second stage. Additionally, we measure the agent's **average cost** (calculated based on token usage and API pricing [3, 20, 45]) and **average steps** per task to assess its efficiency. We describe SRDEL and SRDT below.

SRDEL. We evaluate the metric SRDEL in the first stage:

$$\text{SRDEL} = \frac{\text{\# successful pipelines in data extraction \& loading}}{\text{\# total pipelines}},$$

which measures the proportion of pipelines that successfully extract and load data.

A pipeline is considered successful in the data extraction & loading stage if the pipeline successfully extracts data from all sources and loads it into the data warehouse. To evaluate this, we execute the following query for each source table in the data warehouse:

```
SELECT COUNT(*) FROM source_table;
```

The stage is considered successful only if the size of the loaded data matches the size of the original data.

SRDT. To evaluate the performance of the agent in the second stage, we use the metric SRDT:

$$\text{SRDT} = \frac{\text{\# correctly generated data models}}{\text{\# total data models}},$$

which measures the proportion of correctly generated data models among all data models (one ELT pipeline may involve multiple data models). To assess the correctness of a generated data model, we execute the following query:

```
SELECT * FROM data_model ORDER BY unique_key;
```

The unique key may consist of a composite set of columns determined manually for each data model to ensure the query produces consistent results across different runs. We use this query to create CSV files for the generated data model, which are then compared against the ground truth, which is also derived from the same query.

A generated data model is considered correct if it contains the same number of rows and includes all columns and corresponding values present in the ground truth. Following prior work [38], we allow extra columns in the generated model, as they do not impact functionality. Since the data transformation queries in ELT-Bench require complex logic beyond simply retrieving columns, our metric can accurately reflect the agent's actual performance.

Table 4: ELT-Bench evaluation results for all tested agents with Claude-3.5-Sonnet and GPT-4o. OpenHands CodeActAgent Claude-3.5-Sonnet performs best, with a 73% SRDEL and 11.3% SRDT.

Agent Framework	LLM	SRDEL (%)	SRDT (%)	Average Cost ($)	Average Steps
Spider-Agent [38]	Claude-3.5-Sonnet	23%	0	3.51	63.3
	GPT-4o	15%	0	2.03	43.7
SWE-Agent [76]	Claude-3.5-Sonnet	37%	1%	5.22	60.0
	GPT-4o	0	0	5.22	114.3
Augment Agent [12]	Claude-3.5-Sonnet	45%	2.5%	1.11	50.9
	GPT-4o	1%	0	0.79	30.4
OpenHands CodeActAgent [68, 69]	Claude-3.5-Sonnet	**73%**	**11.3%**	1.41	72.2
	GPT-4o	0	0	1	38.9
Spider-Agent	Claude-3.7-Sonnet w/ extended thinking	57%	3.9%	4.30	89.3

4.2 AI Agent Frameworks

We select four open-source agents from Spider 2.0 [38] leaderboard (Spider-Agent) and SWE-bench [32] leaderboard (SWE-Agent, OpenHands CodeActAgent, and Augment Agent). Spider-Agent is designed for database-related tasks, which allows direct database access. In contrast, the other three agents are designed for address GitHub issues. We run these four agents with five LLMs, including GPT-4o [46], Claude-3.5-Sonnet [4], two open-sourced LLMs (Llama-3.1-405B-Instruct [25], Qwen2.5-Coder-32B-Instruct [53]), and one reasoning model (DeepSeek-R1 [15]). In addition, as a case study aimed at exploring the frontier reasoning model, we also evaluate Spider-Agent Claude-3.7-Sonnet with extended thinking on ELT-Bench. We describe the settings of these four agents below.

Spider-Agent. Spider-Agent is an agent designed for database-related tasks, providing command-line interfaces for multi-turn interactions with environments [38]. It also enables direct interaction with databases to extract detailed source table information (e.g., column values) and verify the correctness of transformation queries (e.g., `DBT` may fail to detect format errors). The agent uses the ReAct [79] framework, in which the LLM generates thought and decides the next action based on current observation and history trajectory at each iteration. We use the default parameter settings of Spider-Agent, except for changing the maximum allowed steps to 100, as ELT-Bench presents more challenging tasks compared to Spider 2.0 [38].

SWE-Agent. SWE-Agent is a code agent designed to address GitHub issues [76]. In each iteration, the agent interacts with the filesystem based on its observations. SWE-Agent operates as a function-calling agent by prompting the LLM to invoke predefined functions, and it also offers a thought-action mode. We use function calling for GPT-4o and Claude-3.5-Sonnet, and the thought-action mode for Llama-3.1-405B-Instruct, Qwen2.5-Coder-32B-Instruct, and DeepSeek-R1 since they fail to call tools correctly. We apply the default parameter settings of SWE-Agent, with one modification: retaining the last 25 observations for the agent due to the complexity of ELT-Bench. Following prior work [76], we allocate a same cost budget to all evaluated LLMs. To establish a comparable budget for both SWE-Agent and Spider-Agent, we first estimate the cost of completing 100 agent steps using Spider-Agent across all LLMs. We then select the highest of these estimates and round it up to the nearest integer, yielding a budget of $6 for each evaluated LLM.

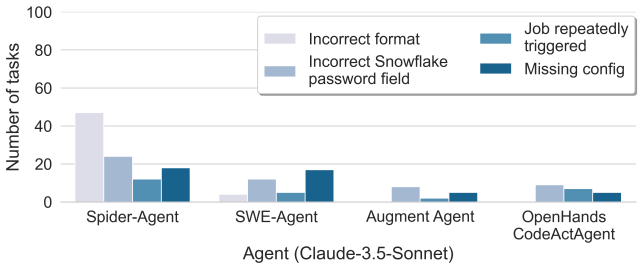
Augment Agent. Augment Agent is a code agent that uses function calling for direct interaction with its execution environment [12]. The agent includes diverse tools, including file viewing, file editing, bash command execution, and sequential thinking. Due to cost limitations, we disable the majority-vote ensemble module, which would otherwise generate multiple candidate solutions and select the best one. We set the maximum allowed steps to 100.

OpenHands CodeActAgent. CodeActAgent is a code agent built on the CodeAct framework, which enables LLMs to generate code as executable actions rather than as plain text or JSON [68]. Following the settings in the SWE-bench leaderboard, we use CodeActAgent within the OpenHands platform [69], a platform for software development agents. OpenHands offers an action for condensing conversation history, thereby maintaining context efficiency throughout extended interactions. We set the maximum allowed steps to 100.

4.3 Evaluation Results

We report our evaluation results for all agents with Claude-3.5-Sonnet and GPT-4o in Table 4. The poor performance, high cost, and extensive action steps highlight the challenges of ELT-Bench. The top-performing agent, OpenHands CodeActAgent Claude-3.5-Sonnet, attains a 73% success rate for data extraction & loading, but only a 11.3% overall success rate. Despite these limitations, this agent demonstrates substantial performance improvements over the second performing agent using a reasoning model, Spider-Agent Claude-3.7-Sonnet with extended thinking, with 28.1% relative improvement in the data extraction and loading stage and 189.7% relative improvement in the data transformation stage. Moreover, ELT-Bench presents a higher computational cost compared to Spider 2.0 [38]. While 30 agent steps are sufficient for most tasks in Spider 2.0, with an average cost of $0.30 per instance using Spider-Agent GPT-4o, evaluating Spider-Agent GPT-4o on ELT-Bench requires an average of 43.7 steps and costs $2.03 per task.

We present a detailed error analysis for the baseline agent evaluations in Section 5, followed by an in-depth case study of Spider-Agent Claude-3.7-Sonnet in Section 6. We provide the performance



(a) Statistics of agents with Claude-3.5-Sonnet.

(b) Statistics of agents with GPT-4o.

Figure 3: Number of tasks failing in Stage 1 categorized by four main issues: incorrect format, incorrect Snowflake password, job repeatedly triggered, and missing configuration for multiple flat files.

```
Action: CreateFile(filepath='/root/.dbt/profiles.yml':
```retail_complains: ...```)
```

(a) Incorrect action format.

```
provider "airbyte" {username = "<username>"}}
```

(b) Incorrect code format.

Figure 4: Incorrect action format generated by Spider-Agent Claude-3.5-Sonnet and incorrect code format generated by SWE-Agent GPT-4o.

of evaluated agents with open-source LLMs and detailed error analysis in the supplementary material [33].

5 ERROR ANALYSIS

In this section, we examine the errors encountered by agents powered by Claude-3.5-Sonnet and GPT-4o. We begin by providing a detailed analysis of errors arising during the data extraction & loading stage. We then investigate the errors occurring in the data transformation stage, focusing on those agents having an SRDEL greater than 10%.

5.1 Error Analysis of Data Extraction & Loading

We examined the major issues encountered by four agents when using GPT-4o and Claude-3.5-Sonnet, including incorrect action or code formats, incorrect `Snowflake` password fields, incorrect table sizes, and missing configuration for multiple flat files. We describe each of these problems in detail below.

Failure to generate action or code in the required format. We observed that LLMs frequently generate actions in incorrect formats when used with Spider-Agent, and similarly produce code in incorrect formats under SWE-Agent. Spider-Agent, which builds on the ReAct framework [79], requires that the LLM produce a valid action at each iteration, terminating the process if three formatting errors are detected via regular expressions. Despite providing detailed descriptions and examples of all actions, LLMs still generate invalid actions. For instance, in Figure 4a, Spider-Agent Claude-3.5-Sonnet incorrectly places a closing parenthesis on the last line

Success case

Action 1: Bash(code="cat /workspace/config.yaml")
Action 10: Bash(code="cat /workspace/documentation/destination_snowflake.md")
Action 11: EditFile(filepath="/workspace/elt/main.tf")
```resource "airbyte_destination_snowflake" "snowflake" {
  configuration = {credentials = {
    username_and_password = {password = "Snowflake@123"}}}}
 (Omitted other fields)```
**Action 14:** Bash(code="terraform apply -auto-approve")
...

**(a) The execution path of a successful task in Stage 1. The agent writes the configuration after reading the documentation.**

**Failure case**

**Action 1:** Bash(code="cat /workspace/config.yaml")
**Action 6:** EditFile(filepath="/workspace/elt/main.tf")
```resource "airbyte_destination" "snowflake" {
password  = "Snowflake@123"}
 (Omitted other fields)```
**Action 8:** Bash(code="terraform apply -auto-approve")
**Observation 8:** The provider airbytehq/airbyte does not support resource type "airbyte_destination"
**Action 14:**Bash(code="cat /workspace/documentation/destination_snowflake.md")
```resource "airbyte_destination_snowflake"  "snowflake"{
    password  = "Snowflake@123"}
(Omitted other fields)```
...

**(b) The execution path of a failed task in Stage 1. The agent writes the configuration file before reading the documentation and only fixes the detected error after reading the documentation.**

**Figure 5: Comprasion the execution path of a successful task and a failed task.**

instead of before the colon in the first line, causing a parsing failure. As shown in Figure 3, Spider-Agent Claude-3.5-Sonnet terminates

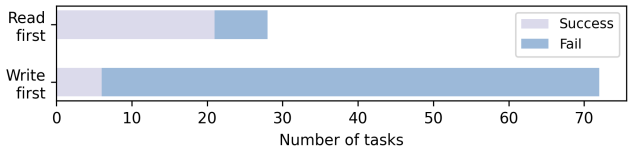



**Figure 6: The success and failure rates of Spider-Agent GPT-4o in Stage 1 under two strategies: when reading the documentation first (27% of tasks), it achieves a 78.8% success rate; when writing the configuration first (73% of tasks), the success rate drops to 9.6%.**

47% of tasks in Stage 1 because of unparsable actions, while Spider-Agent GPT-4o only terminates 7% of tasks.

SWE-Agent uses a function-calling framework with well-defined functions, which avoids parsing errors. However, misformatted code can still be generated. For instance, SWE-Agent GPT-4o generates an extra right curly brace at the end of a code block, as illustrated in Figure 4. We show in Figure 3 that SWE-Agent Claude-3.5-Sonnet produces misformatted code in 4% of cases, whereas SWE-Agent GPT-4o exhibits a 28% error rate.

In contrast, both OpenHands CodeActAgent and Augment Agent are instructed to verify file modifications subsequent to editing. As shown in Figure 3, only OpenHands CodeActAgent GPT-4o produced code with formatting errors in eight tasks. All other agents successfully generated code conforming to the expected formatting standards across all evaluated tasks.

**Failure to configure the `Snowflake` password field.** We examined the `Snowflake` password field, which must be written in the format as shown in Figure 5a. However, as Figure 3 illustrates, agents demonstrate a failure rate of 8% to 98% in configuring the `Snowflake` password field. In addition, for the same agent, using Claude-3.5-Sonnet reduces the failure rate by 49% to 89% compared to GPT-4o. This is because GPT-4o is trained on an outdated version of the `Airbyte Terraform` documentation.

We further analyzed the reasons behind the ineffectiveness of the provided documentation by analyzing the execution path of Spider-Agent GPT-4o. We identified two distinct strategies the agent adopted when configuring `Airbyte Terraform`. In one strategy (Figure 5a), the agent attempts to write the configuration code first and then runs `terraform apply -auto-approve`. Upon encountering an error indicating an incorrect resource type, the agent consults the documentation but only corrects the specific issue reported by `Terraform`. Because `Airbyte Terraform` ignores any fields that are not explicitly defined, other misconfigurations remain undetected, which finally causes the ELT pipeline to fail.

In contrast, when the agent references the documentation before writing the configuration, it is more likely to produce a valid `Terraform` configuration, leading to a higher success rate for data extraction & loading. As illustrated in Figure 6, the agent reads the documentation before writing the configuration in 27 tasks and successfully configures the `Snowflake` password field in 21 tasks. By comparison, in 73 tasks, the agent writes the configuration first, and only six tasks succeed. These observations underscore the importance of the agent's effective planning (e.g., executing actions in the correct sequence) in achieving higher success rates.

**Incorrect loaded table size due to the synchronization job being repeatedly triggered.** We observed that, in some cases, the size of the loaded tables did not match the size of the original data. Analyzing the execution paths of failed cases, we found that the agent repeatedly triggered the same synchronization job. For example, if the original dataset contains 100 rows but the agent executes the synchronization job three times, the loaded table in `Snowflake` ends up with 300 rows instead of the intended 100. As shown in Figure 3, the issue of repeated synchronization job triggers in up to 12 separate tasks when running agents with Claude-3.5-Sonnet. When using GPT-4o, the majority of tasks fail before triggering the synchronization job. These findings highlight the importance of short-term memorization in the agent for tracking executed actions and preventing redundant synchronization jobs.

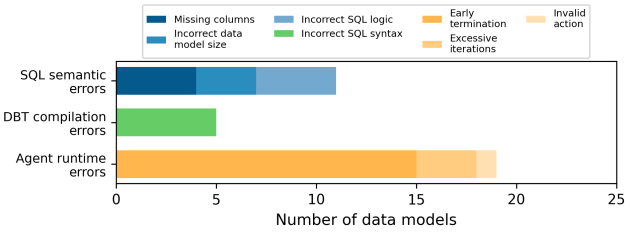
**Missing configuration when having multiple flat files.** For `Postgres`, `MongoDB`, APIs, and `Amazon S3`, multiple tables or files can be configured within a single source block and a single connection block. In contrast, `Airbyte Terraform` requires individual source and connection configuration blocks for each flat file. ELT-Bench includes 24 instances to evaluate whether the agent can correctly generate multiple configuration blocks when having multiple flat files. As illustrated in Figure 3, among the 24 tasks that involve multiple flat files, OpenHands CodeActAgent Claude-3.5-Sonnet and Augment Agent Claude-3.5-Sonnet fail to configure some flat files in 5 tasks, while the worst-performing agent (SWE-Agent GPT-4o) fail in 24 tasks.

## 5.2 Error Analysis of Data Transformation

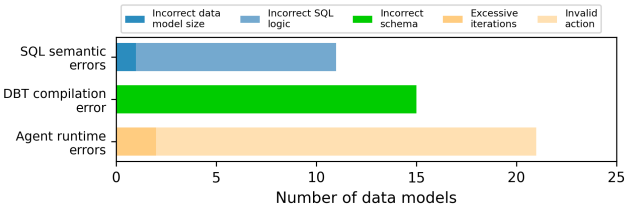
We categorize Stage 2 major errors into three main types: agent runtime errors, `DBT` compilation errors, and SQL semantic errors.

**Agent runtime errors.** Agent runtime errors are defined as failures in which an agent does not successfully generate a data model within the data warehouse. These errors arise from four primary causes: inefficiency, early termination, invalid actions, and prompt length limitations.

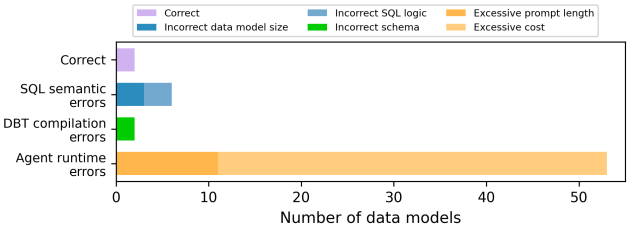
(1) Inefficiency occurs when an agent exceeds the allocated computational budget or step limit without completing the task. As illustrated in Figure 7, 2.6% to 66.7% of attempted data models fail due to inefficiency.
(2) Early termination refers to scenarios where the agent erroneously concludes that the task is complete despite having not fulfilled all requirements; for instance, 42.9% of data models in Spider-Agent GPT-4o occurred due to early termination.
(3) Invalid actions arise when the agent generates an action that is syntactically or semantically incorrect, resulting in exceptions that halt execution. We found that 71.1% of data models in Augment Agent Claude-3.5-Sonnet failed due to invalid actions.
(4) Prompt length limitations were encountered when the agent invoked an incorrect `Airbyte API` call, which resulted in excessively long responses. This caused 17.5% of data models in SWE-Agent Claude-3.5-Sonnet to fail due to exceeding the maximum context length of Claude-3.5-Sonnet.



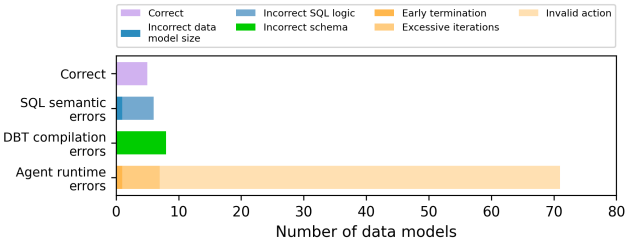

(a) Statistics of Spider-Agent GPT-4o.



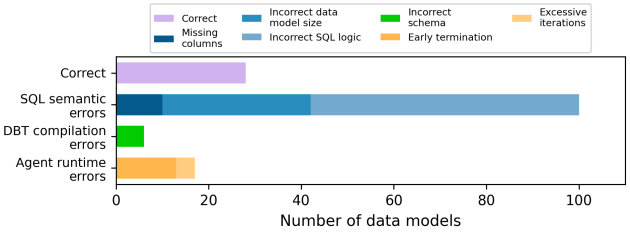

(b) Statistics of Spider-Agent Claude-3.5-Sonnet.




(c) Statistics of SWE-Agent Claude-3.5-Sonnet.



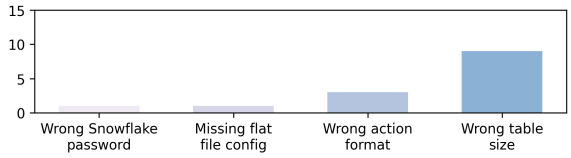

(d) Statistics of Augment Agent Claude-3.5-Sonnet.



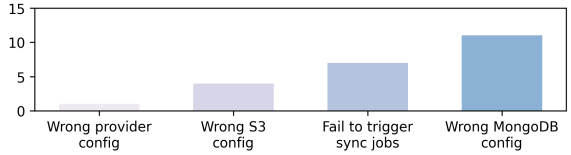

(e) Statistics of OpenHands CodeActAgent Claude-3.5-Sonnet.

**Figure 7: Statistics of agent performance on generating data models Stage 2. Each subfigure includes results for databases that the agent successfully completed in the first stage (35, 47, 63, 90, and 151 data models, respectively).**

**DBT compilation errors.** The second category of errors involves DBT compilation errors, which can be further divided into two types. The first type arises from incorrect DBT configurations, causing data models to be materialized in unintended database schemas. As shown in Figure 7b, Spider-Agent Claude-3.5-Sonnet misplaces 33.3% of its data models in incorrect schemas. The second type results from the generation of transformation queries that contain syntax errors. For example, as illustrated in Figure 7a, Spider-Agent GPT-4o produces transformation queries with syntax errors in 14.3% of data models.




(a) Error types in Section 5.1.




(b) Other common error types.

**Figure 8: Distribution of error types for Spider-Agent Claude 3.7 Sonnet in Stage 1.**

**SQL semantic errors.** SQL semantic errors refers to cases where the agent generates an incorrect data model within the data warehouse, highlighting the limitations of text-to-SQL capabilities. We categorize these errors according to their severity, prioritizing missing columns over incorrect data model size and, subsequently, flawed SQL logic. For example, if a generated data model both omits required columns and exhibits an incorrect total number of rows, the error is classified as a missing columns issue. As illustrated in Figure 7e, SQL semantic errors constitute the most prevalent failure mode in OpenHands CodeActAgent Claude-3.5-Sonnet, affecting 66.2% of data models. Within this category, 10% of the errors correspond to missing columns, 32% to incorrect data model size, and 58% to flawed SQL logic.

## 6 CASE STUDY

In this section, we present an in-depth analysis of the Spider-Agent Claude-3.7-Sonnet with extended thinking, focusing on its performance and the errors encountered across two stages of the task. We then examine its action trajectories in successful cases.

Spider-Agent Claude-3.7-Sonnet achieves a 57% success rate in SRDEL, a 34% improvement compared to Spider-Agent Claude-3.5-Sonnet. It also achieves partial success on 34% of tasks, meaning it loads some required data sources but not all within a task. We further analyzed common error types during the first stage, with results depicted in Figure 8. Our examination of the four issue types described in Section 5.1 shows that Spider-Agent Claude-3.7-Sonnet significantly reduces error frequencies across all categories compared to Spider-Agent Claude-3.5-Sonnet, achieving up to a 95.8% error reduction. We further examined additional common

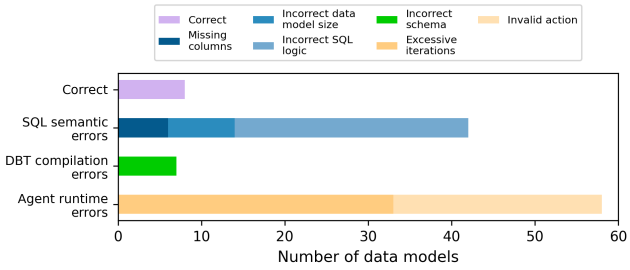



**Figure 9: Statistics of Spider-Agent Claude-3.7-Sonnet in the second stage.**

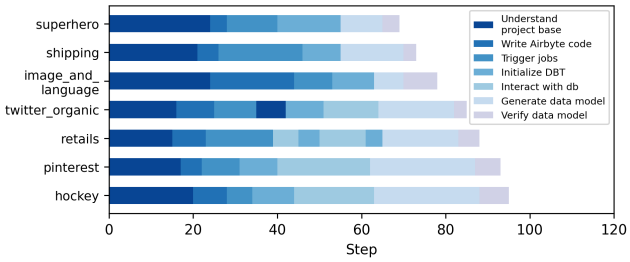



**Figure 10: The action trajectories of the agent on databases with at least one successful data model.**

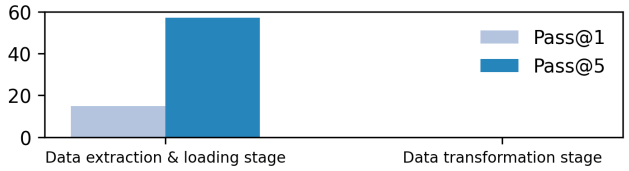



**Figure 11: The success rate of Spider-Agent GPT-4o with one versus five attempts. The success rate improves from 15% to 57% in the first stage but remains 0% in the second stage.**

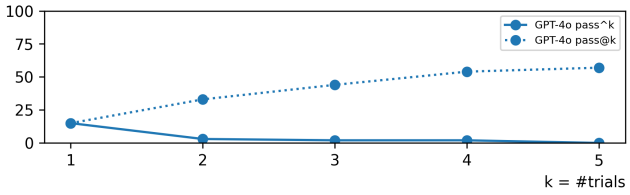



**Figure 12: Pass^k and pass@k in the first stage of ELT-Bench.**

issues of Spider-Agent Claude-3.7-Sonnet. As shown in Figure 8b, Spider-Agent Claude-3.7-Sonnet frequently fails on specific data source types, particularly `MongoDB`.

Spider-Agent Claude-3.7-Sonnet demonstrates a 3.9% performance improvement in the second stage compared to Spider-Agent Claude-3.5-Sonnet. As illustrated in Figure 9, the primary issues of Spider-Agent Claude-3.7-Sonnet in the second stage include excessive iterations (28.7%), incorrect SQL logic (24.3%), and invalid actions (21.7%).

To better understand Spider-Agent Claude-3.7-Sonnet's workflow, we illustrate the action paths of the agent for databases that successfully produced at least one correct data model in Figure 10. On average, the agent executed 83.6 steps for each successful case. To provide clarity, we categorize these actions into defined phases based on the agent's thoughts and actions. Specifically, if fewer than five consecutive steps belonging to one phase appear between two occurrences of another identical phase, we group these intermediate steps into the surrounding phase. For instance, it is common for the agent to briefly interact with the database during the "generate data model" phase. As depicted in Figure 10, the Spider-Agent Claude-3.7-Sonnet spends most of its execution steps to the phases of "understanding the project base" (averaging 20.6 steps) and "generating the data model" (averaging 17 steps).

# 7 SENSITIVITY AND ABLATION STUDY

## 7.1 Multiple Runs Improve Agent Performance

We evaluated Spider-Agent GPT-4o's performance on ELT-Bench with one attempt (pass@1) and five attempts (pass@5). As shown in Figure 11, Spider-Agent GPT-4o achieves a pass@5 rate of 57% in Stage 1, indicating that in 57% of tasks, at least one of the five attempts successfully extracts data from multiple sources and loads it into the data warehouse. This result represents a 3.8× improvement over its pass@1 performance. However, in Stage 2, despite having more successfully loaded source tables, Spider-Agent GPT-4o still fails to build a correct data model.

We further use the pass^k metric [77] to evaluate the consistency and robustness of Spider-Agent GPT-4o on ELT-Bench. As shown in Figure 12, as the number of trials increases, pass^k for Spider-Agent GPT-4o drops significantly, eventually reaching 0 when k equals 5, indicating the need for a more robust agent in future work.

## 7.2 Using Documentation Improves Agent Performance

We evaluated whether Spider-Agent Claude-3.5-Sonnet and Spider-Agent GPT-4o could complete the data extraction & loading stage without consulting documentation. Since LLMs are trained on a fixed knowledge cutoff, their ability to reference up-to-date documentation is crucial for completing real-world tasks. To assess their adaptability, we compared their performance in data extraction & loading with and without documentation guidance.

In our experiments, we provided the agents with documentation on configuring `Airbyte` `Terraform` and invoking the `Airbyte` API to initiate synchronization jobs. As shown in Figure 13, Spider-Agent Claude-3.5-Sonnet and Spider-Agent GPT-4o exhibit degraded performance in the data extraction & loading stage when documentation is unavailable. Without access to documentation, Spider-Agent Claude-3.5-Sonnet succeeds in only one task, while Spider-Agent GPT-4o fails in all tasks. These findings reveal that both Claude-3.5-Sonnet and GPT-4o rely not only on memorized knowledge but also on their reasoning abilities to complete tasks.

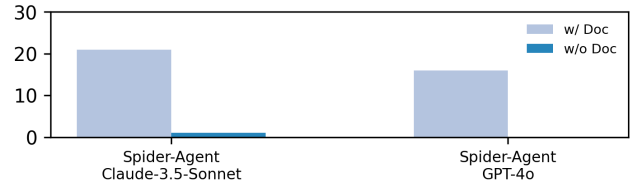



**Figure 13: The success rate of Spider-Agent with Claude-3.5-Sonnet and GPT-4o in the data extraction & loading stage, evaluated with and without documentation. Success rates decrease from 21% to 1% for Claude-3.5-Sonnet, and from 15% to 0% for GPT-4o.**

**Table 5: Evaluation results for all agents on the isolated data transformation stage. OpenHands CodeActAgent Claude-3.5-Sonnet achieves the highest success rate at 15.8%.**

| Agent Framework | LLM | SRDT (%) |
|---|---|---|
| Spider-Agent | Claude-3.5-Sonnet | 3.5% |
| | GPT-4o | 2.0% |
| | DeepSeek-R1 | 0 |
| | Llama-3.1-405B-Instruct | 0 |
| | Qwen2.5-Coder-32B-Instruct | 0 |
| SWE-Agent | Claude-3.5-Sonnet | 4.9% |
| | GPT-4o | 1.5% |
| | DeepSeek-R1 | 2.0% |
| | Llama-3.1-405B-Instruct | 0 |
| | Qwen2.5-Coder-32B-Instruct | 0 |
| Augment Agent | Claude-3.5-Sonnet | 2.0% |
| | GPT-4o | 0 |
| | Llama-3.1-405B-Instruct | 0 |
| Openhands CodeactAgent | Claude-3.5-Sonnet | **15.8%** |
| | GPT-4o | 1.0% |
| | DeepSeek-R1 | 4.9% |
| | Llama-3.1-405B-Instruct | 0 |
| | Qwen2.5-Coder-32B-Instruct | 0.5% |
| Spider-Agent | Claude-3.7-Sonnet w/ extended thinking | 11.8% |

## 7.3 Performance on Isolated Data Transformation

We evaluated four agents during the data transformation stage, given that the source tables had already been loaded into the data warehouse. Furthermore, we assessed the performance of a text-to-SQL system, MAC-SQL [67], to generate transformation queries based on the target data model's schema.

**Agent performance on isolated data transformation stage.** We evaluated agent performance on the isolated data transformation stage, with source tables already loaded into the data warehouse. Each agent was provided with both the target data model schema and the source table schemas. Agents were tasked with configuring the corresponding DBT project—specifically, generating the profile file and writing transformation queries to produce the target data model. As shown in Table 5, Openhands CodeactAgent Claude-3.5-Sonnet achieves the highest success rate at 15.8%. We compared the performance of different LLMs using the same agent, OpenHands CodeActAgent. As shown in Figure 14, OpenHands CodeActAgent with Claude-3.5-Sonnet achieves the best performance, surpassing the others by a margin of 10.9% to 15.8%. Additionally, we compared the performance of different agents with Claude-3.5-Sonnet. As shown in Figure 15, OpenHands CodeActAgent outperforms other agents by 10.9% to 13.8%.

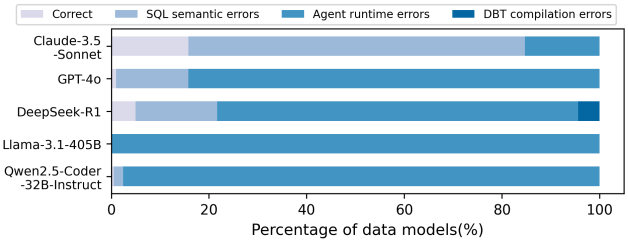



**Figure 14: Distribution of data model generation results using OpenHands CodeActAgent with different LLMs. Running with Claude-3.5-Sonnet can improve the performance from 10.9% to 15.8%.**

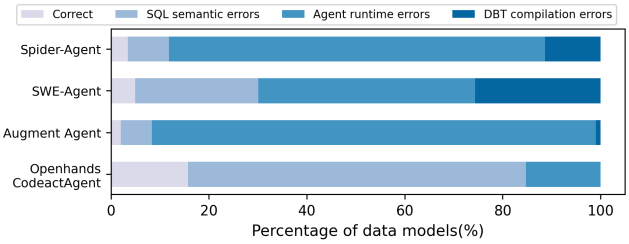



**Figure 15: Distribution of data model generation results using four agents with Claude-3.5-Sonnet. Openhands CodeActAgent can improve the performance from 10.9% to 13.8%.**

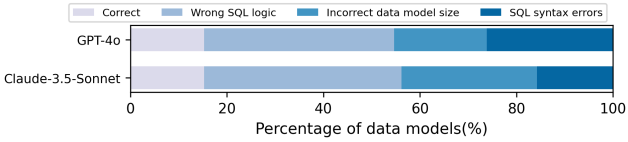



**Figure 16: Distribution of data model generation results using MAC-SQL. Both MAC-SQL GPT-4o and MAC-SQL Claude-3.5-Sonnet successfully generate 15.3% of data models.**

**Text-to-SQL system performance on data transformation query generation.** We also evaluated a text-to-SQL system, MAC-SQL [67], which includes a selector, decomposer, and refiner modules. The data transformation query generation task is to generate the SQL query based on the target data model's schema and the source tables' schema. We ran MAC-SQL with GPT-4o and

Claude-3.5-Sonnet. As shown in Figure 16, both MAC-SQL GPT-4o and MAC-SQL Claude-3.5-Sonnet achieve a 15.3% success rate. We further analyzed the failure reasons of MAC-SQL GPT-4o and MAC-SQL Claude-3.5-Sonnet. As illustrated in Figure 16, MAC-SQL GPT-4o fails to generate valid data models due to syntax errors in 26.1% of cases, and produces data models with incorrect or missing columns in 39.4% of cases. By comparison, MAC-SQL Claude-3.5-Sonnet exhibits a lower syntax error rate at 15.8%, while generating data models with incorrect or missing columns in 40.9% of cases.

## 8 RELATED WORK

**ELT and ETL data pipelines.** ELT and ETL data pipelines are essential for converting raw data into structured, reliable formats, playing an important role in modern data engineering workflows. ETL techniques have been extensively studied over decades [60], while the rise of cloud data warehousing has driven the increasing adoption of ELT pipelines [22, 43, 56, 61]. In ETL workflows, transformations are often handled by a secondary processing server using different languages like Java, Python, or Scala, while in ELT workflows, transformations are performed within the data warehouse using SQL. Early research mainly focus on conceptual modeling for ETL processes [42, 63, 65]. More recent efforts have aimed at automating various stages of ETL and ELT pipelines to minimize engineering effort, including Semantic Web-based approaches for attribute mapping [62], template-driven automatic data loading [10], and machine learning-based data integration [44]. Since the increasing adoption of ELT pipelines, we introduce ELT-Bench, a benchmark designed to facilitate the development of AI agents that automate ELT pipeline construction, thus reducing manual effort.

**Benchmarks for data systems and AI agents.** We review related benchmarks for data systems and AI agents. For data systems, the TPC benchmark suite represents a standard line of work, evaluating system throughput across various scenarios. Notable benchmarks include TPC-DI [48], targeting data integration workloads; TPCx-AI [7], designed for AI and machine learning systems; and TPCx-BB [1], focused on big data analytics. Complementary to these are text-to-SQL benchmarks [19, 31, 40, 75, 81, 83], which evaluate the ability of systems to generate SQL queries from natural language questions. Recent efforts have expanded the scope of these benchmarks: TAG-Bench[6] assesses a system's capability to answer analytical questions requiring LLM-driven inference over database contents, such as semantic reasoning and world-knowledge augmentation. TQA-Bench [52] assesses the capability of LLMs in multi-table question answering. These two benchmarks can still be regarded as data-analytic benchmarks and do not include ELT operations.

Beyond database-oriented benchmarks, researchers have developed diverse benchmarks to assess AI agent performance in broader domains, including software engineering [32], machine learning [29], and web-based interaction environments [17, 84]. At the intersection of data systems and AI agents, Spider 2-V [9] evaluates agent proficiency in using data tools, while Spider 2.0 [38] focuses on enterprise-oriented text-to-SQL tasks. In this work, we introduce ELT-Bench, a benchmark that covers the entire data engineering workflow. ELT-Bench is designed to assess the capabilities of AI agents in constructing real-world, end-to-end ELT pipelines.

**Text-to-SQL benchmarks and methods.** Researchers have studied the text-to-SQL task for decades. Initially, text-to-SQL methods primarily leverage graph neural networks (GNNs) [8] and long short-term memory (LSTM) networks[74]. Recent research has increasingly adopted fine-tuning techniques [24, 39] and prompting approaches [16, 23, 50] to further enhance SQL generation accuracy with the advent of LLMs. ELT-Bench tasks agents with generating complex SQL transformation queries to construct data models based on provided column names and descriptions. These queries typically involve intricate structures, including nested subqueries and multi-table joins.

**LLM-powered systems and AI agents.** AI Agents have emerged as a promising approach for addressing real-world challenges across various fields, including software engineering [68, 76, 82], web browsing [36, 47], and data science and engineering [27, 28, 38]. These agents typically consist of four crucial modules: reasoning [35, 70, 78], tool usage [51, 55], planning [59, 80], and memorization [85]. Furthermore, recent efforts have focused on developing LLM-powered systems for data processing tasks, including Pneuma for tabular data representation and retrieval [5], and DocETL for processing complex documents [57]. The task presented in ELT-Bench exemplifies a standard data engineering workflow for processing structured data. We leverage ELT-Bench to evaluate the effectiveness of four AI agents in building ELT pipelines.

## 9 CONCLUSION

We introduce ELT-Bench, a comprehensive end-to-end benchmark specifically designed for real-world ELT pipeline tasks in the data engineering domain. ELT-Bench aims to replicate realistic scenarios by providing environments for diverse data sources and integrating widely adopted data tools. The benchmark presents a substantial challenge, as the top-performing agent, OpenHands CodeActAgent Claude-3.5-Sonnet, correctly generates data models in only 11.3% of cases. This performance gap highlights significant opportunities for future research to develop more powerful and intelligent AI agents capable of handling complex ELT workflows.

## 10 LIMITATIONS AND FUTURE WORK

ELT-Bench evaluates AI agent performance in constructing ELT pipelines. However, our current implementation assumes that the input data is pre-cleaned. In real-world scenarios, raw data often contains inconsistencies, errors, or missing values. Future work can consider extending ELT-Bench with a data cleaning and preprocessing stage. This extension could be implemented using SQL-based transformations for declarative and scalable cleaning operations, or via script-based approaches (e.g., using Python or other languages) to handle more complex preprocessing tasks.

## 11 ACKNOWLEDGEMENTS

This work was supported in part by Google. We are grateful to the CloudLab [18] for providing computing resources for experiments.

# REFERENCES


[1] Dippy Aggarwal, Shreyas Shekhar, Chris Elford, Umachandar Jayachandran, Sadashivan Krishnamurthy, Jamie Reding, and Brendan Niebruegge. 2019. TPCx-BB (Big Bench) in a Single-Node Environment. In *Performance Evaluation and Benchmarking for the Era of Cloud(s): 11th TPC Technology Conference, TPCTC 2019, Los Angeles, CA, USA, August 26, 2019, Revised Selected Papers* (Los Angeles, CA, USA). Springer-Verlag, Berlin, Heidelberg, 64–83. https://doi.org/10.1007/978-3-030-55024-0_5

[2] Airbyte. 2025. Airbyte. https://airbyte.com/

[3] Anthropic. [n.d.]. Anthropic API Pricing. https://www.anthropic.com/pricing#anthropic-api

[4] Anthropic. 2024. The Claude 3 Model Family: Opus, Sonnet, Haiku. https://api.semanticscholar.org/CorpusID:268232499

[5] Muhammad Imam Luthfi Balaka, David Alexander, Qiming Wang, Yue Gong, Adila Krisnadhi, and Raul Castro Fernandez. 2025. Pneuma: Leveraging LLMs for Tabular Data Representation and Retrieval in an End-to-End System. *Proceedings of the ACM on Management of Data* 3, 3 (June 2025), 1–28. https://doi.org/10.1145/3725337

[6] Asim Biswal, Liana Patel, Siddarth Jha, Amog Kamsetty, Shu Liu, Joseph E. Gonzalez, Carlos Guestrin, and Matei Zaharia. 2024. Text2SQL is Not Enough: Unifying AI and Databases with TAG. arXiv:2408.14717 [cs.DB] https://arxiv.org/abs/2408.14717

[7] Christoph Brücke, Philipp Härtling, Rodrigo D Escobar Palacios, Hamesh Patel, and Tilmann Rabl. 2023. TPCx-AI - An Industry Standard Benchmark for Artificial Intelligence and Machine Learning Systems. *Proc. VLDB Endow.* 16, 12 (Aug. 2023), 3649–3661. https://doi.org/10.14778/3611540.3611554

[8] Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. LGESQL: Line Graph Enhanced Text-to-SQL Model with Mixed Local and Non-Local Relations. arXiv:2106.01093 [cs.CL] https://arxiv.org/abs/2106.01093

[9] Ruisheng Cao, Fangyu Lei, Haoyuan Wu, Jixuan Chen, Yeqiao Fu, Hongcheng Gao, Xinzhuang Xiong, Hanchong Zhang, Yuchen Mao, Wenjing Hu, Tianbao Xie, Hongshen Xu, Danyang Zhang, Sida Wang, Ruoxi Sun, Pengcheng Yin, Caiming Xiong, Ansong Ni, Qian Liu, Victor Zhong, Lu Chen, Kai Yu, and Tao Yu. 2024. Spider2-V: How Far Are Multimodal Agents From Automating Data Science and Engineering Workflows? arXiv:2407.10956 [cs.AI] https://arxiv.org/abs/2407.10956

[10] Malu Castellanos, Alkis Simitsis, Kevin Wilkinson, and Umeshwar Dayal. 2009. Automating the loading of business process data warehouses. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology* (Saint Petersburg, Russia) *(EDBT '09)*. Association for Computing Machinery, New York, NY, USA, 612–623. https://doi.org/10.1145/1516360.1516431

[11] Abhirup Chatterjee and Arie Segev. 1991. Data manipulation in heterogeneous databases. *SIGMOD Rec.* 20, 4 (Dec. 1991), 64–68. https://doi.org/10.1145/141356.141385

[12] Augment Code. 2025. Augment SWE-bench Verified Agent. https://github.com/augmentcode/augment-swebench-agent

[13] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. 2016. The Snowflake Elastic Data Warehouse. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 215–226. https://doi.org/10.1145/2882903.2903741

[14] dbt Labs. 2025. dbt. https://www.getdbt.com/

[15] DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948 [cs.CL] https://arxiv.org/abs/2501.12948

[16] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. 2023. C3: Zero-shot Text-to-SQL with ChatGPT. arXiv:2307.07306 [cs.CL] https://arxiv.org/abs/2307.07306

[17] Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. 2024. WorkArena: How Capable Are Web Agents at Solving Common Knowledge Work Tasks? arXiv:2403.07718 [cs.LG] https://arxiv.org/abs/2403.07718

[18] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 1–14. https://www.usenix.org/conference/atc19/presentation/duplyakin

[19] Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving Text-to-SQL Evaluation Methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 351–360. https://doi.org/10.18653/v1/p18-1033

[20] Fireworks. [n.d.]. Fireworks Model Library. https://fireworks.ai/models

[21] Fivetran. 2025. Fivetran. https://github.com/fivetran

[22] Harald Foidl, Valentina Golendukhina, Rudolf Ramler, and Michael Felderer. 2024. Data pipeline quality: Influencing factors, root causes of data-related issues, and processing problem areas for developers. *Journal of Systems and Software* 207 (2024), 111855. https://doi.org/10.1016/j.jss.2023.111855

[23] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proc. VLDB Endow.* 17, 5 (Jan. 2024), 1132–1145. https://doi.org/10.14778/3641204.3641221

[24] Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, Jinyang Gao, Liyu Mou, and Yu Li. 2025. A Preview of XiYan-SQL: A Multi-Generator Ensemble Framework for Text-to-SQL. arXiv:2411.08599 [cs.AI] https://arxiv.org/abs/2411.08599

[25] Aaron Grattafiori and et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[26] Mauricio A. Hernández and Salvatore J. Stolfo. 1998. Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Min. Knowl. Discov.* 2, 1 (Jan. 1998), 9–37. https://doi.org/10.1023/A:1009761603038

[27] Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Chenxing Wei, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Xiangru Tang, Xiangtao Lu, Xiawu Zheng, Xinbing Liang, Yaying Fei, Yuheng Cheng, Zhibin Gou, Zongze Xu, and Chenglin Wu. 2024. Data Interpreter: An LLM Agent For Data Science. arXiv:2402.18679 [cs.AI] https://arxiv.org/abs/2402.18679

[28] Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, Yao Cheng, Jianbo Yuan, Jiwei Li, Kun Kuang, Yang Yang, Hongxia Yang, and Fei Wu. 2024. InfiAgent-DABench: Evaluating Agents on Data Analysis Tasks. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (Eds.), Vol. 235. PMLR, 19544–19572. https://proceedings.mlr.press/v235/hu24s.html

[29] Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2024. MLAgentBench: Evaluating Language Agents on Machine Learning Experimentation. arXiv:2310.03302 [cs.LG] https://arxiv.org/abs/2310.03302

[30] Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, and Kang Liu. 2024. DA-Code: Agent Data Science Code Generation Benchmark for Large Language Models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 13487–13521. https://doi.org/10.18653/v1/2024.emnlp-main.748

[31] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a Neural Semantic Parser from User Feedback. arXiv:1704.08760 [cs.CL] https://arxiv.org/abs/1704.08760

[32] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? arXiv:2310.06770 [cs.CL] https://arxiv.org/abs/2310.06770

[33] Tengjun Jin, Yuxuan Zhu, and Daniel Kang. 2025. ELT-Bench: An End-to-End Benchmark for Evaluating AI Agents on ELT Pipelines (Supplementary Material). https://github.com/uiuc-kang-lab/ELT-Bench/blob/eltbench/materials/supplemental_material.pdf

[34] Ralph Kimball and Joe Caserta. 2004. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming and Delivering Data.* John Wiley & Sons, Inc., Hoboken, NJ, USA.

[35] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large Language Models are Zero-Shot Reasoners. arXiv:2205.11916 [cs.CL] https://arxiv.org/abs/2205.11916

[36] Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. 2024. AutoWebGLM: A Large Language Model-based Web Navigating Agent. arXiv:2404.03648 [cs.CL] https://arxiv.org/abs/2404.03648

[37] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Scott Wen tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2022. DS-1000: A Natural and Reliable Benchmark for Data Science Code Generation. arXiv:2211.11501 [cs.SE] https://arxiv.org/abs/2211.11501

[38] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2024. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. arXiv:2411.07763 [cs.CL] https://arxiv.org/abs/2411.07763

[39] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. CodeS: Towards Building Open-source Language Models for Text-to-SQL. arXiv:2402.16347 [cs.CL] https:

//arxiv.org/abs/2402.16347

[40] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C.C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2024. Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-SQLs. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 1835, 28 pages.

[41] LocalStack. 2025. LocalStack: A Fully Functional Local AWS Cloud Stack. https://github.com/localstack/localstack

[42] Sergio Luján-Mora, Panos Vassiliadis, and Juan Trujillo. 2004. Data Mapping Diagrams for Data Warehouse Design with UML, Vol. 3288. 191–204. https://doi.org/10.1007/978-3-540-30464-7_16

[43] Anthony Mbata, Yaji Sripada, and Mingjun Zhong. 2024. A Survey of Pipeline Tools for Data Engineering. arXiv:2406.08335 [cs.LG] https://arxiv.org/abs/2406.08335

[44] Kartick Chandra Mondal, Neepa Biswas, and Swati Saha. 2020. Role of Machine Learning in ETL Automation. In *Proceedings of the 21st International Conference on Distributed Computing and Networking* (Kolkata, India) *(ICDCN '20)*. Association for Computing Machinery, New York, NY, USA, Article 57, 6 pages. https://doi.org/10.1145/3369740.3372778

[45] OpenAI. [n.d.]. OpenAI API Pricing. https://openai.com/api/pricing

[46] OpenAI. 2024. GPT-4o System Card. arXiv:2410.21276 [cs.CL] https://arxiv.org/abs/2410.21276

[47] Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. 2024. Autonomous Evaluation and Refinement of Digital Agents. arXiv:2404.06474 [cs.AI] https://arxiv.org/abs/2404.06474

[48] Meikel Poess, Tilmann Rabl, Hans-Arno Jacobsen, and Brian Caufield. 2014. TPC-DI: the first industry benchmark for data integration. *Proc. VLDB Endow.* 7, 13 (Aug. 2014), 1367–1378. https://doi.org/10.14778/2733004.2733009

[49] Daniel Poppy. 2023. ETL vs ELT: What's the difference? https://www.getdbt.com/blog/etl-vs-elt

[50] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. arXiv:2304.11015 [cs.CL] https://arxiv.org/abs/2304.11015

[51] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. arXiv:2307.16789 [cs.AI] https://arxiv.org/abs/2307.16789

[52] Zipeng Qiu, You Peng, Guangxin He, Binhang Yuan, and Chen Wang. 2024. TQA-Bench: Evaluating LLMs for Multi-Table Question Answering with Scalable Context and Symbolic Extension. arXiv:2411.19504 [cs.AI] https://arxiv.org/abs/2411.19504

[53] Qwen. 2025. Qwen2.5 Technical Report. arXiv:2412.15115 [cs.CL] https://arxiv.org/abs/2412.15115

[54] Raza Rasool and Ali Afzal Malik. 2015. Effort estimation of ETL projects using Forward Stepwise Regression. In *2015 International Conference on Emerging Technologies (ICET)*. 1–6. https://doi.org/10.1109/ICET.2015.7389209

[55] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. arXiv:2302.04761 [cs.CL] https://arxiv.org/abs/2302.04761

[56] Dhamotharan Seenivasan. 2022. ETL vs ELT: Choosing the right approach for your data warehouse. *International Journal for Research Trends and Innovation* (2022), 110–122.

[57] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G. Parameswaran, and Eugene Wu. 2025. DocETL: Agentic Query Rewriting and Evaluation for Complex Document Processing. arXiv:2410.12189 [cs.DB] https://arxiv.org/abs/2410.12189

[58] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv:2303.11366 [cs.AI] https://arxiv.org/abs/2303.11366

[59] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv:2303.11366 [cs.AI] https://arxiv.org/abs/2303.11366

[60] Alkis Simitsis, Spiros Skiadopoulos, and Panos Vassiliadis. 2023. The History, Present, and Future of ETL Technology (invited). In *Proceedings of the 25th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP) co-located with the 26th International Conference on Extending Database Technology and the 26th International Conference on Database Theory (EDBT/ICDT 2023), Ioannina, Greece, March 28, 2023 (CEUR Workshop Proceedings)*, Enrico Gallinucci and Lukasz Golab (Eds.), Vol. 3369. CEUR-WS.org, 3–12. https://ceur-ws.org/Vol-3369/invited1.pdf

[61] Bharat Singhal and Alok Aggarwal. 2022. ETL, ELT and reverse ETL: a business case Study. In *2022 Second International Conference on Advanced Technologies in Intelligent Control, Environment, Computing & Communication Engineering (ICATIECE)*. IEEE, 1–4.

[62] Dimitrios Skoutas and Alkis Simitsis. 2006. Designing ETL processes using semantic web technologies. In *Proceedings of the 9th ACM International Workshop on Data Warehousing and OLAP* (Arlington, Virginia, USA) *(DOLAP '06)*. Association for Computing Machinery, New York, NY, USA, 67–74. https://doi.org/10.1145/1183512.1183526

[63] Juan Trujillo and Sergio Luján-Mora. 2003. A UML Based Approach for Modeling ETL Processes in Data Warehouses, Vol. 2813. 307–320. https://doi.org/10.1007/978-3-540-39648-2_25

[64] Alexander van Renen and Viktor Leis. 2023. Cloud Analytics Benchmark. *Proc. VLDB Endow.* 16, 6 (Feb. 2023), 1413–1425. https://doi.org/10.14778/3583140.3583156

[65] Panos Vassiliadis, Alkis Simitsis, and Spiros Skiadopoulos. 2002. Conceptual modeling for ETL processes. In *Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP* (McLean, Virginia, USA) *(DOLAP '02)*. Association for Computing Machinery, New York, NY, USA, 14–21. https://doi.org/10.1145/583890.583893

[66] Florian Waas, Robert Wrembel, Tobias Freudenreich, Maik Thiele, Christian Koncilia, and Pedro Furtado. 2013. On-Demand ELT Architecture for Right-Time BI: Extending the Vision. *International Journal of Data Warehousing and Mining* 9 (04 2013), 21–38. https://doi.org/10.4018/jdwm.2013040102

[67] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. arXiv:2312.11242 [cs.CL] https://arxiv.org/abs/2312.11242

[68] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable Code Actions Elicit Better LLM Agents. arXiv:2402.01030 [cs.CL] https://arxiv.org/abs/2402.01030

[69] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. 2025. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=OJd3ayDDoF

[70] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL] https://arxiv.org/abs/2201.11903

[71] Lilian Weng. 2023. LLM-powered Autonomous Agents. *lilianweng.github.io* (Jun 2023). https://lilianweng.github.io/posts/2023-06-23-agent/

[72] Jennifer Widom. 1995. Research problems in data warehousing. In *Proceedings of the Fourth International Conference on Information and Knowledge Management* (Baltimore, Maryland, USA) *(CIKM '95)*. Association for Computing Machinery, New York, NY, USA, 25–30. https://doi.org/10.1145/221270.221319

[73] Niklas Wretblad, Fredrik Gordh Riseby, Rahul Biswas, Amin Ahmadi, and Oskar Holmström. 2024. Understanding the Effects of Noise in Text-to-SQL: An Examination of the BIRD-Bench Benchmark. arXiv:2402.12243 [cs.CL] https://arxiv.org/abs/2402.12243

[74] Xiaojun Xu, Chang Liu, and Dawn Song. 2017. SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning. arXiv:1711.04436 [cs.CL] https://arxiv.org/abs/1711.04436

[75] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. SQLizer: query synthesis from natural language. *Proc. ACM Program. Lang.* 1, OOPSLA, Article 63 (Oct. 2017), 26 pages. https://doi.org/10.1145/3133887

[76] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. arXiv:2405.15793 [cs.SE] https://arxiv.org/abs/2405.15793

[77] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. $\tau$-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. arXiv:2406.12045 [cs.AI] https://arxiv.org/abs/2406.12045

[78] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. arXiv:2305.10601 [cs.CL] https://arxiv.org/abs/2305.10601

[79] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL] https://arxiv.org/abs/2210.03629

[80] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL] https://arxiv.org/abs/2210.03629

[81] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2019. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. arXiv:1809.08887 [cs.CL] https://arxiv.org/abs/1809.08887

[82] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024. AutoCodeRover: Autonomous Program Improvement. arXiv:2404.05427 [cs.SE] https://arxiv.org/abs/2404.05427
[83] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. arXiv:1709.00103 [cs.CL] https://arxiv.org/abs/1709.00103
[84] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. WebArena: A Realistic Web Environment for Building Autonomous Agents. arXiv:2307.13854 [cs.AI] https://arxiv.org/abs/2307.13854
[85] Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, Yu Qiao, Zhaoxiang Zhang, and Jifeng Dai. 2023. Ghost in the Minecraft: Generally Capable Agents for Open-World Environments via Large Language Models with Text-based Knowledge and Memory. arXiv:2305.17144 [cs.AI] https://arxiv.org/abs/2305.17144