Paper DH03

Enhancing Clinical Trial Data Queries with LLMs and Neo4j: A Flexible Framework for ADaM Dataset Management

Jaime Yan, Merck & Co., Inc., Rahway, NJ, USA
Changhong Shi, Merck & Co., Inc., Rahway, NJ, USA

ABSTRACT

CDISC ADaM standards are essential for clinical trial data management, ensuring regulatory alignment and data consistency. While traditional SQL-based querying methods remain prevalent, graph databases offer unique advantages for handling interconnected clinical trial data. This study presents an innovative framework that leverages Neo4j graph database for ADaM datasets, using Large Language Models (LLMs) to generate Cypher queries while benchmarking against conventional SQL approaches. We developed a reusable graph schema specifically designed for ADaM datasets, enabling efficient storage and retrieval of clinical trial data within Neo4j. The framework allows users to input natural language questions, which are processed through a graph similarity search to identify relevant schema elements. These elements serve as input to the LLM, which primarily generates Cypher queries while maintaining SQL compatibility for comparative analysis. Our results demonstrate the relative strengths of graph-based querying compared to traditional SQL approaches, specifically in terms of execution time, accuracy, and resource utilization for clinical trial data management tasks. This framework makes data checking more flexible, particularly for ad hoc tasks, by simplifying the querying process through Cypher while providing SQL-based performance metrics as a reference point.

1 INTRODUCTION

1.1 Background

CDISC ADaM standards are an essential component of clinical trial data management and analysis. These standards are crucial for aligning data with Food and Drug Administration guidelines, supporting the efficient generation, replication, and review of statistical analyses [1]. The principles and rules of CDISC ADaM are designed to be flexible, allowing adaptation to various trials across different therapeutic areas and companies. In the industry, specifications in Excel format are widely used to contain schema information. These specifications help

standardize the structure and content of clinical trial datasets, facilitating data consistency and regulatory compliance.

Data querying is a critical aspect of managing and analyzing clinical trial datasets. Efficient data querying techniques enhance the ability to retrieve and manipulate data, supporting more accurate and comprehensive analyses. Luthria and Wang [2] highlighted the importance of developing innovative data-sharing methods to protect patient privacy while meeting the needs of clinical trial investigators. Gursoy, Brannon, and Gerstein [3] introduced a solution using Ethereum blockchain to store and query pharmacogenomics data efficiently, emphasizing the importance of high-integrity data storage and quick access.

Recent advancements have shown the integration of large language models (LLMs) with SQL queries to be promising. Saeed, De Cao, and Papotti [4] demonstrated the potential of using SQL queries to tap into information stored in LLMs, expanding the scope of data that can be queried beyond traditional databases. Additionally, interactive systems like DExPlorer enable users to discover and rank desired tuples without writing SQL queries [5]. The development of SQL synthesizers, such as SQUARES, which utilizes query reverse engineering, further simplifies the process for users [6].

The use of LLMs in graph data query processing has also gained significant attention. Tian et al. [7] introduced IBM Db2 Graph, which integrates graph queries within the IBM Db2 relational database using a graph overlay technique. Furthermore, Zhong et al. [8] proposed SyntheT2C, a methodology for generating synthetic Query-Cypher pairs for fine-tuning LLMs on the Text2Cypher task, utilizing LLM-based prompting and template-filling techniques. These advancements illustrate the potential of LLMs in enhancing the querying capabilities for both relational and graph databases.

1.2 Motivation and Our Approach

Despite these advancements, there is a gap in utilizing LLMs for generating SQL and Cypher queries specifically for clinical trial data. Existing methods focus on protecting data privacy and efficient data storage but do not address the need for user-friendly query generation systems that can understand and translate natural language questions into effective database queries. Additionally, there is a lack of comparative studies on the performance of SQL and Cypher queries in answering clinical trial-related questions.

Our proposed framework aims to fill this gap by leveraging an LLM to generate both SQL and Cypher queries from natural language questions. This approach is motivated by the need for a more intuitive and accessible querying system that can facilitate data retrieval and analysis for clinical researchers. By comparing the performance of these queries, we seek to provide insights into the most efficient and practical querying methods for clinical trial data.

Our approach involves generating synthetic clinical data based on CDISC ADaM specifications in Excel format and preparing SQL schema for a PostgreSQL database and graph schema for a Neo4j database. We import `.sas7bdat` format data into both SQL and Graph databases and utilize LangChain with different LLMs to build a query system capable of understanding and translating natural language questions into SQL and Cypher queries. To provide a comprehensive evaluation, we created a benchmark consisting of 40 questions extracted from study protocols using the graph RAG technique. This benchmark is used to test and compare the performance of SQL and Cypher queries generated by the LLM in

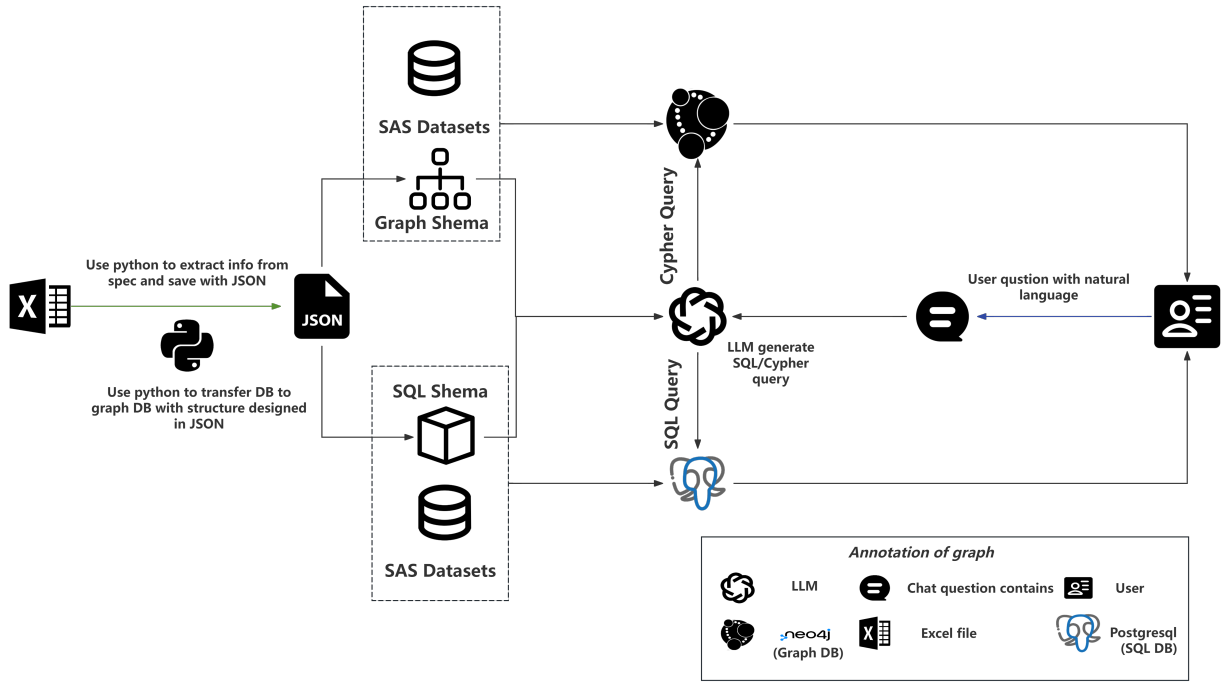

Figure 1: Flow of the framework.

terms of execution time, accuracy, and resource utilization.

This study distinguishes itself from previous work by focusing on the practical application of LLMs for query generation in clinical trial data management, especially for CDSIC ADaM style clinical trial data. Unlike previous methods, our framework provides a direct comparison between SQL and Cypher queries, offering valuable insights into their respective efficiencies and practicalities in answering clinical trial-related questions.

2 DATA STRUCTURES AND QUERY LANGUAGES

Clinical trial data management requires a structured and standardized approach to ensure data consistency, regulatory compliance, and comprehensive analysis. The Clinical Data Interchange Standards Consortium (CDISC) provides the Analysis Data Model (ADaM) Implementation Guide (IG) as a manual to define the structure and content for each dataset. These datasets are meticulously designed to capture specific aspects of a subject or patient, resulting in high-dimensional data due to the extensive information recorded for each participant.

The ADaM IG specifies several key datasets, each serving a distinct purpose:

- **ADSL (Subject-Level Analysis Dataset)**: Captures demographic and baseline variables.
- **ADAE (Adverse Event Analysis Dataset)**: Records adverse event-related variables.

- **ADCM (Concomitant Medication Dataset)**: Contains data on concomitant medications.
- **ADLB (Laboratory Test Results Dataset)**: Includes laboratory test records.
- **ADVS (Vital Signs Dataset)**: Documents vital signs data.

Each dataset provides a detailed perspective on the subjects, which presents a unique challenge: the need to transform these diverse datasets into a cohesive schema that can be queried efficiently. The industry typically addresses this by using specifications (spec) files in Excel format. These files save critical details such as variable formats, lengths, derivation information, data structures, key variables, and standardized code/decode lists for clinical items (e.g., country codes, laboratory test codes).

The initial challenge lies in converting these specs into SQL and graph schemas while preserving the integrity of the dataset structures and inter-dataset relationships. This involves several steps:

1. **Extraction of Dataset Structure and Key Variables**: Identifying the structure and key variables for each dataset.
2. **Variable Information Compilation**: Extracting variable details such as labels and types, and integrating code lists to provide code/decode information.
3. **JSON Conversion**: Organizing the extracted information hierarchically in a JSON format, ensuring clarity and ease of processing by large language models (LLMs).

The primary objective is to transfer the separate SAS datasets, along with their specifications, into well-defined schemas and databases. This includes creating a SQL schema for a relational database (such as PostgreSQL) and a graph schema for a Neo4j database.

Extracted Jason structure from Spec

```
{
  "Dataset Name": {
    "Structure": "str",
    "Key Variables": "str",
    "Variables": [
      {
        "VARIABLE": "str",
        "LABEL": "str",
        "DATA TYPE": "str",
        "Codelist": "NoneType",
        "Code": "str",
        "Terms": [
          "..."
        ]
      }
    ]
  }
}
```

In our case study, we focus on the datasets pertinent to safety analysis, specifically ADSL, ADLB, ADAE, ADCM, and ADVS. By converting the specs into a JSON structure, we create a foundation that supports both SQL and graph schema development. This JSON format maintains the integrity of the variable information and the intricate relationships between datasets, facilitating accurate and efficient data analysis.

The subsequent sections will detail the processes for transforming this JSON structure into SQL and graph schemas, and then importing the data into the respective databases. We will illustrate typical queries used in clinical trial data analysis, demonstrating how both relational and graph databases can be effectively utilized to manage and analyze clinical trial data.

2.1 SQL Schema and Database

In a relational database, clinical trial data adhering to the CDISC ADaM standards is organized into tables with predefined columns and data types. Each table represents a specific aspect of the clinical trial, such as patient demographics, treatment information, and adverse events. The structure is designed to facilitate data normalization, reduce redundancy, and ensure data integrity.

Below is an example for the creating of SQL schema:

SQL schema

```
class ADSL(Base):
    __tablename__ = 'adsl'

    # Core identifiers
    STUDYID = Column(String, comment='Study Identifier')
    USUBJID = Column(String, primary_key=True, comment='
       Unique Subject Identifier')
    SUBJID = Column(String, comment='Subject Identifier for
       the Study')
    SITEID = Column(String, comment='Study Site Identifier')

    # Demographic information
    AGE = Column(Integer, comment='Age')
    AGEU = Column(String, comment='Age Units')
    AGEGR1 = Column(String, comment='Pooled Age Group 1')
    SEX = Column(String, comment='Sex')
    ...Continue
```

Using PostgreSQL, we create the SQL database by importing the .sas7bdat format datasets and applying the created SQL schema as input. This ensures that the relational database is accurately structured to reflect the specifications and relationships defined in the ADaM IG.

Key Points for Creating the SQL Schema:

- Define each dataset separately, ensuring all variable names, formats, and labels are clearly specified.

- Set USUBJID as the primary key for the ADSL dataset to ensure each subject is uniquely identified.
- For other datasets, set USUBJID as a foreign key to establish relationships between the datasets.
- Create a derived ID variable for each record in non-ADSL datasets and set it as the primary key to ensure each record is uniquely identified.
- Use the specifications (spec) files in Excel format to accurately convert and maintain variable details, data structures, and relationships in the SQL schema.

2.2 Graph Schema

In a graph database, clinical trial data is represented as nodes and relationships, providing a flexible and intuitive way to model complex data structures. Nodes represent entities such as patients, treatments, and adverse events, while relationships capture the connections between these entities. This model is particularly useful for exploring relationships and patterns within the data.

Based on the JSON file, we utilize Cypher queries to build the graph schema for the graph database. Due to the flexibility of graph databases, we can include all the information contained in the JSON file. While the SQL schema focuses on variables and relationships (primary keys and foreign keys), it does not include the possible values for each variable, especially those with code/decode values. Graph schemas, however, can link code/decode values and incorporate conditions to appropriately classify data into nodes.

For example, the following Cypher query merges sex categories and links them to subjects while setting additional properties:

Cypher to create GraphDB

```
def create_adsl_graph(tx, row):
    query = """
    // Create Study node
    MERGE (s:Study {STUDYID: $STUDYID})

    // Create Site node
    MERGE (site:Site {SITEID: $SITEID})

    // Create Subject node
    MERGE (subj:Subject {USUBJID: $USUBJID})
    SET subj += $subject_props

    // Create relationships
    MERGE (subj)-[:PARTICIPATED_IN]->(s)
    MERGE (subj)-[:ASSIGNED_TO]->(site)

    // Create category nodes with predefined terms/code/
       decode pairs
    MERGE (age:AgeGroup {value: $AGEGR1})
```

```
ON CREATE SET age.order = CASE $AGEGR1
    WHEN '<= 35 years old' THEN 1
    WHEN '> 35 years old' THEN 2
    ELSE null
END
MERGE (subj)-[:HAS_AGE_GROUP]->(age)

MERGE (sex:SexCategory {value: $SEX})
ON CREATE SET sex.order = CASE $SEX
    WHEN 'F' THEN 1
    ELSE null
END
MERGE (subj)-[:HAS_SEX]->(sex)
...Continue
```

This approach allows the graph schema to contain more detailed information compared to the SQL schema . After defining the graph schema, we use Neo4j to import the .sas7bdat datasets into the graph database. This process ensures that the graph database not only captures the relationships between datasets but also includes the specific values and conditions associated with each variable.

Key Points for Creating the Graph Schema:

- Utilize the JSON file to define nodes and relationships in the graph schema.
- Include all variable details, possible values, and conditions to classify data into appropriate nodes.
- Use Cypher queries to merge nodes and set properties based on the values and conditions from the JSON file.
- Import the .sas7bdat datasets into Neo4j using the created graph schema.

3 ROLE OF LLM IN GENERATING QUERIES

3.1 How an LLM Can Be Used to Generate SQL and Cypher Queries

The use of Large Language Models (LLMs) for generating SQL and Cypher queries represents a significant advancement in natural language processing and data querying. LLMs can understand and process natural language inputs, translating them into structured query languages such as SQL and Cypher. This capability is particularly useful in scenarios where users, who may not be proficient in query languages, need to interact with databases to retrieve and analyze data.

Generating SQL Queries:

LLMs can be trained or fine-tuned on datasets containing natural language questions and their corresponding SQL queries. This training enables the model to learn the patterns and structures necessary to translate natural language inputs into accurate SQL queries. The process typically involves the following steps:

1. **Input Processing**: The LLM processes the natural language input to understand the user's query.
2. **Pattern Recognition**: The model recognizes patterns in the input that correspond to specific SQL query components (e.g., SELECT, FROM, WHERE clauses).
3. **Query Generation**: The model generates the SQL query based on the recognized patterns and the structure of the database schema.

For instance, recent research introduced M-SQL, a neural network architecture for single-table text-to-SQL generation, leveraging pre-trained models like BERT to translate natural language queries into SQL statements [9]. Additionally, surveys of various neural network models for text-to-SQL conversion highlight advancements in the field, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), pointer networks, and generative models [10].

Generating Cypher Queries:

Similarly, LLMs can be trained to generate Cypher queries for graph databases. The process includes:

1. **Input Processing**: The LLM processes the natural language input to understand the user's intent.
2. **Node and Relationship Mapping**: The model identifies the nodes and relationships described in the input.
3. **Query Construction**: The model constructs a Cypher query that accurately represents the user's request, including any specific conditions or constraints.

For example, LLMs can convert a natural language query like "Find all patients who have experienced adverse events" into the following Cypher query:

```
MATCH (p:Patient)-[:EXPERIENCED]->(ae:Adverse_Event)
RETURN p
```

Recent advancements in this area include the development of methodologies for generating synthetic data to fine-tune LLMs on the text-to-Cypher task, improving the accuracy and relevance of Cypher queries generated from natural language inputs [8].

Using LangChain for Query Generation:

In our framework, we use LangChain's provided query chains to handle the interaction between user questions and both SQL and graph databases. LangChain offers a variety of tools and templates to facilitate the generation and execution of SQL and Cypher queries. Specifically, the LangChain CypherQAChain component allows us to interact with a Neo4j graph database using natural language. This component translates user questions into Cypher queries, executes them against the graph, and uses the returned context information to generate a natural language response [11, 12].

LangChain facilitates the entire process by:

- Converting natural language inputs into SQL or Cypher queries.
- Executing the generated queries against the respective databases.
- Retrieving and processing the query results.
- Generating a natural language response based on the retrieved data.

3.2 Comparison of Manually Written vs. LLM-Generated Queries

Manually Written Queries:

Manually written queries require users to have a deep understanding of the query language and the database schema. This process can be time-consuming and prone to errors, especially for complex queries or users who are not proficient in the query language.

LLM-Generated Queries:

LLM-generated queries, on the other hand, allow users to interact with databases using natural language. This approach offers several advantages:

- **Accessibility**: Users without expertise in SQL or Cypher can easily retrieve data using natural language inputs.
- **Efficiency**: The time required to formulate and execute queries is significantly reduced.
- **Consistency**: LLMs can generate consistent and accurate queries, reducing the likelihood of errors.

However, it is important to note that LLM-generated queries may sometimes require validation and refinement to ensure they meet specific requirements or constraints.

By leveraging LLMs for query generation and incorporating tools like LangChain, we can enhance the accessibility and efficiency of data retrieval processes, enabling a wider range of users to interact with complex databases effectively.

4 PERFORMANCE METRICS

4.1 Accuracy and Relevance of Query Results

Accuracy measures how correctly the query results match the expected outcomes, while relevance pertains to how well the query results meet the user's needs or the intended analysis purpose. Ensuring that queries return accurate and relevant results is fundamental for reliable data analysis. In the context of clinical trials, accuracy directly impacts the validity of conclusions drawn from the data.

Evaluating Accuracy and Relevance of Query Results:

To evaluate the accuracy and relevance of query results, we utilize a Retrieval-Augmented Generation (RAG) approach based on protocol files to generate benchmark questions. RAG models combine the strengths of traditional information retrieval techniques with sequence-to-sequence models to enhance question-answering frameworks [13, 14]. This method allows us to generate relevant and accurate benchmark questions from clinical trial protocols. These

questions are then used to test the results from both SQL query chains and Cypher query chains.

We compare the results of LLM-generated queries with expected results using validation datasets. This involves running the queries on known datasets where the outcomes are already established. To quantify the accuracy and relevance of query outputs, we implement precision, recall, and F1-score metrics. Precision measures the proportion of true positive results among the total positive results returned by the query. Recall measures the proportion of true positive results out of the actual positives. F1-score is the harmonic mean of precision and recall, providing a single metric that balances both concerns.

Recent advancements in RAG models have shown significant improvements in generating domain-specific answers using fewer parameters and leveraging massive knowledge graphs [15, 16]. These models ensure that the generated answers are grounded in relevant knowledge and aligned with the context of the questions being asked.

By implementing these performance metrics, we can comprehensively evaluate and enhance the accuracy and relevance of LLM-generated queries in clinical trial data analysis. This structured approach ensures that the data retrieval processes are both effective and reliable, ultimately supporting better decision-making in clinical research.

5 METHODOLOGY

5.1 Data Preparation

The data preparation phase involves three main steps: input data acquisition, Excel processing and JSON schema creation, and synthetic dataset generation. This process transforms raw clinical trial specifications into a structured format suitable for further analysis and modeling.

5.1.1 Input Data

The initial inputs for this study comprise two primary sources:

- ADAM (Analysis Data Model) Specifications: These adhere to the CDISC ADaM standards and include key datasets such as:
  - ADSL (Subject-Level Analysis Dataset)
  - ADAE (Adverse Events Analysis Dataset)
  - ADCM (Concomitant Medications Analysis Dataset)
  - ADLB (Laboratory Test Results Analysis Dataset)
  - ADVS (Vital Signs Analysis Dataset)
- Corresponding Clinical Trial Protocol

The ADAM specifications are provided in Excel format, containing critical details such as variable formats, lengths, derivation information, data structures, key variables, and standardized code/decode lists.

The clinical trial protocol is a comprehensive document that outlines the study's objectives, design, methodology, statistical considerations, and organization. It typically includes:

5.1.2 Excel Processing and JSON Schema Creation

To facilitate easier data analysis and integration, we transform the ADAM specifications from Excel format into a structured JSON schema. This process, illustrated in Figure 2, involves several steps:

1. Loading the Excel file containing ADAM specifications
2. Extracting data from three key sheets: Datasets, Variables, and Codelists
3. Renaming columns for consistency and extracting required information
4. Merging data from Datasets and Variables sheets
5. Processing and splitting Codelists into dataset and variable components
6. Merging processed data with the main dataset, handling variable-only matches
7. Aggregating and simplifying the data structure
8. Creating a hierarchical JSON schema organized by dataset, including:
    - Dataset structure
    - Key variables
    - List of variables with their respective labels, data types, codelists (if applicable), and associated terms
9. Saving the final simplified and aggregated data as a JSON file

This JSON schema provides a clear, hierarchical representation of the ADAM specifications, making it more accessible for further analysis and easier for Language Models to process.

5.1.3 Synthetic Dataset Generation

Due to the sensitive nature of clinical trial data, which often contains private patient information and is proprietary to clinical trial support companies, we generate synthetic datasets for this study. The synthetic data generation process, detailed in our companion paper [citation], involves the following steps:

1. Utilizing the JSON schema derived from ADAM specifications
2. Incorporating relevant information from the clinical trial protocol
3. Employing Large Language Models (LLMs) to interpret the schema and protocol
4. Generating Faker code based on the interpreted specifications
5. Using the Faker package to create synthetic datasets with .sas7bdat

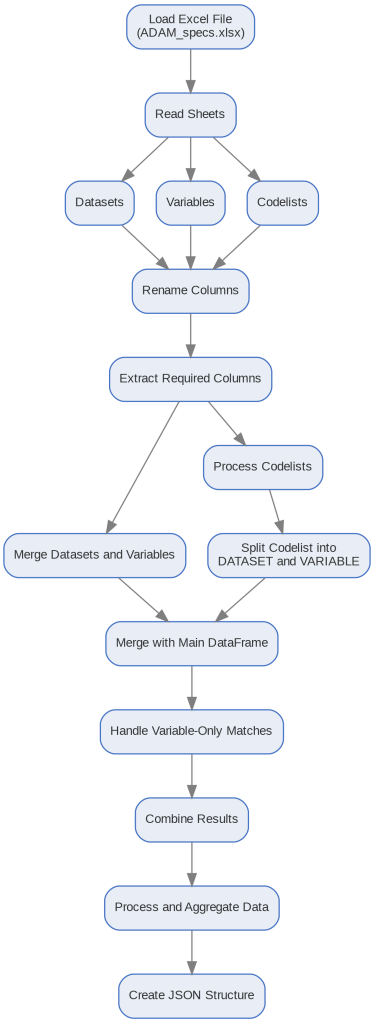

Figure 2: Flow chart for excel file processing.

The LLMs analyze the JSON schema and protocol to understand the required data structure, variable types, and constraints. They then generate Python code using the Faker library, a package designed to produce realistic fake data. This Faker code is tailored to create synthetic data that mimics the characteristics of real clinical trial data while adhering to the specific requirements outlined in the ADAM specifications and protocol. For example, the LLM might generate Faker code to create age distributions that match the study's inclusion criteria, or to produce adverse event frequencies that align with those expected in the study population. The Faker package then executes this code to generate the final synthetic datasets.

This approach ensures that our synthetic data strictly adheres to ADaM standards while protecting patient privacy and intellectual property. The resulting datasets accurately reflect the structure, distributions, and relationships found in real clinical trial data, allowing for robust analysis and model development without compromising confidentiality.

By using this method, we can generate large volumes of realistic, yet entirely fictional, clinical trial data. This enables thorough testing and validation of our graph-based analysis methods while maintaining the highest standards of data privacy and ethical research practices.

5.2 SQL Schema and SQL Database Creation

The transformation of ADAM specifications into a relational database structure involves several steps, ensuring data integrity and efficient querying capabilities. This process is crucial for maintaining the complex relationships inherent in clinical trial data.

5.2.1 Schema Definition

Each ADAM dataset was defined separately with comprehensive specifications:

- Variable names: Adhering to CDISC naming conventions
- Data formats: Specifying appropriate SQL data types (e.g., VARCHAR, INTEGER, DATE)
- Labels: Providing clear, descriptive labels for each variable
- Constraints: Implementing primary keys, foreign keys, and other constraints to maintain data integrity

The Subject-Level Analysis Dataset (ADSL) serves as the cornerstone of the database structure:

- Primary Key: `USUBJID` (Unique Subject Identifier)
- Contains one record per subject in the study
- Includes key demographic and study conduct variables

Other datasets (e.g., ADAE, ADCM, ADLB, ADVS) are structured as follows:

- Foreign Key: `USUBJID`, linking to ADSL
- Primary Key: A derived ID variable unique to each dataset (e.g., `AESEQ` for ADAE)
- May contain multiple records per subject

5.2.2 JSON to SQL Schema Conversion

The JSON structure created in the earlier data preparation step was programmatically converted into SQL schema definitions:

- A custom Python script was developed to parse the JSON structure
- SQL Data Definition Language (DDL) statements were generated for each dataset
- The script accounted for data types, constraints, and relationships between tables

5.2.3 Database Implementation

We chose PostgreSQL as our relational database management system due to its robust features, performance, and compatibility with clinical data analysis tools. The implementation process involved:

1. Setting up a PostgreSQL server with appropriate security configurations
2. Creating a dedicated database for the clinical trial data
3. Executing the generated SQL DDL statements to create the table structures
4. Importing the synthetic datasets from `.sas7bdat` format into the corresponding SQL tables

The data import process utilized the following tools and techniques:

- SAS Universal ODBC driver: To read `.sas7bdat` files
- Python libraries such as pandas and sqlalchemy: For efficient data manipulation and database interaction
- Bulk insert operations: To optimize the import process for large datasets

5.2.4 Data Validation and Quality Checks

After the database population, we performed several validation steps to ensure data integrity and consistency:

- Row count verification: Comparing the number of records in each SAS dataset with the corresponding SQL table
- Data type checks: Ensuring all variables were correctly typed in the SQL schema
- Constraint validation: Verifying that all primary key and foreign key constraints were satisfied
- Sample data comparison: Manually comparing a subset of records between SAS and SQL to ensure accurate data transfer

5.2.5 Database Optimization

To enhance query performance and data retrieval efficiency, we implemented several optimization techniques:

- Indexing: Created appropriate indexes on frequently queried columns and foreign keys
- Partitioning: For larger tables, implemented table partitioning based on relevant criteria (e.g., study site or date ranges)

- Query optimization: Analyzed and optimized common query patterns using PostgreSQL's EXPLAIN ANALYZE feature

This comprehensive approach to creating and populating the SQL database ensures that the synthetic clinical trial data is stored in a structured, efficient, and easily queryable format. The resulting database serves as a robust foundation for subsequent graph-based analyses and comparisons with traditional relational database approaches.

5.3 GRAPH SCHEMA AND GRAPH DATABASE

5.3.1 Graph Schema Definition

The JSON file derived from the ADAM specifications was used as the foundation for defining the graph schema:

- Nodes:
  - Subject nodes: Representing individual patients
  - Event nodes: Representing various clinical events (e.g., adverse events, lab tests)
  - Category nodes: Representing categorical variables (e.g., sex, age group)
- Relationships:
  - Subject-to-Event: Linking subjects to their clinical events
  - Subject-to-Category: Associating subjects with their categorical attributes
  - Event-to-Category: Connecting events to relevant categories (e.g., severity of adverse events)
- Properties:
  - Node properties: Attributes specific to each node type
  - Relationship properties: Additional information about the connections

5.3.2 Cypher Query Generation

Cypher queries were programmatically generated to create and populate the graph database:

- Node creation queries:

```
MERGE (s:Subject {USUBJID: $USUBJID})
SET s += $subject_properties
```

- Property setting queries:

```
SET subj += $subject_props
```

- Relationship creation queries:

```
MERGE (subj)-[:PARTICIPATED_IN]->(s)
MERGE (subj)-[:ASSIGNED_TO]->(site)
```

- Category node creation with conditional ordering:

```
MERGE (age:AgeGroup {value: $AGEGR1})
ON CREATE SET age.order = CASE $AGEGR1
    WHEN '<= 35 years old' THEN 1
    WHEN '> 35 years old' THEN 2
    ELSE null
END
MERGE (subj)-[:HAS_AGE_GROUP]->(age)
```

- Complex category node creation (e.g., treatment arms):

```
MERGE (arm:ArmCategory {ARMCD: $ARMCD, ARM: $ARM})
ON CREATE SET arm.order = CASE $ARMCD
    WHEN 'A' THEN 1
    WHEN 'SCRNFAIL' THEN 2
    ELSE null
END
MERGE (subj)-[:ASSIGNED_TO_ARM]->(arm)
```

5.3.3 Neo4j Database Creation

Neo4j was employed to create and populate the graph database:

1. Database setup:
    - Installation and configuration of Neo4j
    - Creation of a new graph database instance
2. Schema application:
    - Execution of Cypher queries to define constraints and indexes

- Creation of node labels and relationship types based on the graph schema

3. Data import process:
   - Development of a custom Python script to read `.sas7bdat` files
   - Transformation of tabular data into graph-structured data
   - Batch processing of data to optimize import performance
4. Data validation:
   - Verification of node and relationship counts
   - Sampling of data to ensure correct property values and connections

5.3.4 Graph Database Optimization

To enhance query performance and data retrieval efficiency:

- Indexing:
  - Creation of indexes on frequently queried node properties
  - Implementation of composite indexes for common query patterns
- Query optimization:
  - Analysis of query execution plans using Neo4j's PROFILE command
  - Refactoring of complex queries to improve performance
- Database configuration:
  - Tuning of Neo4j configuration parameters for optimal performance
  - Implementation of appropriate caching strategies

5.4 Query Generation Using LLM

Generating SQL Queries:
Large Language Models (LLMs) were trained or fine-tuned on datasets containing natural language questions paired with corresponding SQL queries. This enabled the models to translate natural language inputs into SQL queries. To facilitate this process, tools like LangChain were utilized, particularly its SQL Database Integration component. This component is responsible for converting user queries into SQL, executing the SQL statements, and retrieving the results from the database.

Generating Cypher Queries:
Similarly, LLMs were trained to generate Cypher queries. The process involved interpreting natural language queries, identifying relevant nodes and relationships within the graph, and constructing the appropriate Cypher queries. LangChain's CypherQAChain component was employed to manage interactions with the Neo4j graph database, converting user inputs into Cypher queries, executing them, and returning the results.

5.5 Evaluation of Query Performance

To evaluate query performance, we employed a Retrieval-Augmented Generation (RAG) approach to generate benchmark questions from clinical trial protocols. RAG models combine traditional information retrieval methods with sequence-to-sequence models to enhance the accuracy and relevance of question-answering frameworks [13, 14]. In this study, we used Chroma as the vector database, with the text-embedding-ada-002 model for embedding, handling protocol documents in DOCX format with a chunk size of 1,000 tokens. We then utilized the gpt-4o-mini model with a prepared prompt to generate 90 questions.

We categorized these benchmark questions into three levels of difficulty: Simple (questions 1-30), Medium (questions 31-60), and Difficult (questions 61-90). For each difficulty level, we generated questions based on topics randomly selected from ADSL, ADAE, ADLB, ADVS, and ADCM datasets. Simple questions involved basic data retrieval and simple calculations from a single dataset, Medium questions required more complex queries, basic statistical analysis, and possibly joins between two datasets, while Difficult questions entailed advanced analysis, multi-step queries, and complex statistics involving multiple datasets.

Domain specialists prepared the correct answers for these questions. The results from the LLM-generated SQL and Cypher queries were then compared with these correct answers to evaluate performance.

Metrics for Evaluation:

- **Precision:** Measures the proportion of true positive results among all positive results returned by the model.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- **Recall:** Measures the proportion of true positive results among all actual positives in the dataset.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- **F1-Score:** The harmonic mean of Precision and Recall, providing a balance between the two metrics.

$$F1\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Recent advancements in RAG models have significantly improved the generation of domain-specific answers using fewer parameters and by leveraging extensive knowledge graphs [15, 16]. These models ensure that the generated responses are well-grounded in relevant knowledge and aligned with the specific context of the questions being asked.

6 RESULTS

Our analysis compared the performance of SQL and Cypher queries across different difficulty levels of clinical trial data queries. Table 1 summarizes the key metrics for both query languages.

The results show that Cypher consistently outperformed SQL across all difficulty levels:

Table 1: Comparison of SQL and Cypher Query Performance Across Difficulty Levels

Difficulty	Metric	SQL			Cypher		
		Accuracy	F1 Score	Correct/Total	Accuracy	F1 Score	Correct/Total
	Simple	0.8000	0.7725	24/30	0.8667	0.8450	26/30
	Medium	0.6667	0.6325	20/30	0.7333	0.7100	22/30
	Difficult	0.5333	0.4950	16/30	0.6000	0.5725	18/30
Overall		0.6667	0.6333	60/90	0.7333	0.7092	66/90

- For simple queries, Cypher achieved an accuracy of 86.67% (F1: 0.8450) compared to SQL's 80.00% (F1: 0.7725).
- In medium-difficulty queries, Cypher maintained a higher accuracy of 73.33% (F1: 0.7100) than SQL's 66.67% (F1: 0.6325).
- For difficult queries, while both languages saw a decrease in performance, Cypher (60.00%, F1: 0.5725) still outperformed SQL (53.33%, F1: 0.4950).
- Overall, Cypher demonstrated superior performance with an accuracy of 73.33% (F1: 0.7092) compared to SQL's 66.67% (F1: 0.6333).

7 DISCUSSION

The analysis of SQL and Cypher query performance in the context of clinical trial data retrieval reveals several important insights:

7.1 Overall Performance

Cypher consistently outperformed SQL across all difficulty levels, with an overall accuracy of 73.33% (F1: 0.7092) compared to SQL's 66.67% (F1: 0.6333). This difference suggests that Cypher may be more suitable for complex data retrieval tasks in clinical trial databases. The lower F1 scores compared to accuracy metrics indicate the presence of partial matches and edge cases in query generation, which is typical in text-to-query systems.

7.2 Performance Across Difficulty Levels

Both query languages showed a decline in performance as query complexity increased, reflected in both accuracy and F1 scores:

- For simple queries, Cypher's performance (86.67% accuracy, F1: 0.8450) was notably higher than SQL (80.00% accuracy, F1: 0.7725). This suggests that Cypher may be particularly effective for routine, straightforward data retrieval tasks, with F1 scores remaining close to accuracy due to fewer edge cases.

- In medium-difficulty queries, while both languages saw a decrease in performance, Cypher (73.33% accuracy, F1: 0.7100) still outperformed SQL (66.67% accuracy, F1: 0.6325). The widening gap between accuracy and F1 scores reflects the increasing complexity of query matching.

- For difficult queries, both languages showed a significant performance drop, with Cypher (60.00% accuracy, F1: 0.5725) maintaining an advantage over SQL (53.33% accuracy, F1: 0.4950). The largest gap between accuracy and F1 scores at this level indicates the challenges in handling complex query conditions and edge cases.

The consistent pattern of lower F1 scores compared to accuracy metrics across all difficulty levels highlights the nuanced nature of query matching in clinical data contexts, where partial matches and varying degrees of correctness are common. This pattern becomes more pronounced as query complexity increases, suggesting that both languages face similar challenges in handling complex clinical data queries, though Cypher maintains a performance advantage throughout.

7.3 Implications for Clinical Trial Data Management

The superior performance of Cypher across all difficulty levels has significant implications for clinical trial data management:

1. **Improved Data Retrieval Accuracy:** The higher accuracy rates of Cypher suggest that it could lead to more reliable and precise data retrieval in clinical trial databases.

2. **Efficiency in Complex Queries:** Cypher's maintained performance advantage in difficult queries indicates its potential for handling complex data relationships often present in clinical trial data.

3. **Learning Curve Considerations:** While Cypher shows better performance, the adoption of a new query language may require additional training for database administrators and researchers familiar with SQL.

4. **Scalability:** As clinical trials grow in complexity and data volume, Cypher's performance advantage could become even more pronounced, potentially offering better scalability for future research needs.

7.4 LLM Performance and Future Improvements

An important observation from our study is the impact of LLM performance on query generation:

- We noticed that some incorrect results were due to errors in SQL and Cypher queries generated by the LLM (gpt-4o-mini).

- These errors led to query execution failures, impacting the overall accuracy of our framework.

- This observation suggests that the performance of our framework is closely tied to the capabilities of the underlying LLM.

As LLM technology continues to advance rapidly, we anticipate significant improvements in query generation accuracy:

- Future iterations of LLMs are likely to have enhanced understanding of database schemas and query languages.
- This could lead to a reduction in syntax errors and logical mistakes in generated queries.
- Consequently, the overall performance of our framework is expected to improve substantially with LLM advancements.

7.5 Limitations and Future Directions

While our study provides valuable insights, it's important to acknowledge its limitations:

- The use of synthetic data, while necessary for this study, may not fully capture the nuances of real-world clinical datasets.
- The performance of LLM-generated queries may vary with different LLM models or training approaches.
- Factors such as database optimization and hardware performance, which can significantly impact query execution in real-world scenarios, were not fully explored.

Future research directions could include:

- Validating these findings with real-world CDISC ADaM datasets from completed clinical trials.
- Exploring the impact of different LLM models and fine-tuning approaches on query generation accuracy.
- Investigating the potential of hybrid approaches that combine the strengths of SQL and Cypher for comprehensive clinical data analysis.
- Assessing the long-term implications of adopting graph databases and Cypher in the clinical trial data management workflow.
- Investigating the impact of different LLM models (e.g., GPT-4, future models) on query generation accuracy and overall framework performance.
- Developing specialized fine-tuning approaches for LLMs to enhance their understanding of CDISC ADaM standards and clinical trial database schemas.

8 CONCLUSION

This study introduces a pioneering framework that employs Large Language Models (LLMs) to generate SQL and Cypher queries, specifically tailored for the analysis of clinical trial data adhering to CDISC ADaM standards. We discovered that Cypher offers a distinct performance advantage over SQL in terms of query accuracy, particularly in handling complex data relationships inherent in clinical trial datasets. This advantage underscores the potential of graph-based querying methods, especially when integrated with LLM-driven natural language interfaces, to significantly enhance the efficiency and accessibility of clinical trial data management.

Cypher's superior handling of complex queries aligns well with the increasing complexity and volume of data in modern clinical trials. By facilitating natural language querying, our framework has the potential to democratize access to clinical trial data analysis, enabling a broader spectrum of researchers and clinicians to engage more effectively with trial data. This could accelerate the pace of clinical research and improve decision-making processes.

Despite these advantages, the implementation of new querying methods like Cypher in the highly regulated field of clinical trials will require careful consideration. Future efforts should aim to validate these findings with real-world datasets, assess the implications for regulatory compliance, and explore the scalability of graph-based methods in large-scale, multi-center trials.

Furthermore, our study acknowledges the limitations associated with the current generation of LLMs, notably the inaccuracies in query generation by gpt-4o-mini that led to execution failures. This highlights the critical role of LLM performance in our framework's effectiveness and points to significant opportunities for improvement as LLM technology continues to advance. We anticipate that future iterations of LLMs will demonstrate improved capabilities in understanding database schemas, query languages, and the nuanced specifics of clinical trial data structures, thereby reducing syntax errors and logical mistakes in generated queries.

In conclusion, while our findings advocate for the utility of LLM-generated Cypher queries in enhancing clinical trial data analysis, they also outline a clear trajectory for future advancements. As LLM technology progresses, the accuracy and reliability of our framework are expected to increase substantially. This progression promises to further amplify the benefits of graph-based querying approaches in clinical research, potentially leading to more efficient data analysis, accelerated drug development processes, and ultimately, improved patient outcomes. Future research should leverage these technological advancements to refine and optimize LLM-based querying systems for clinical trial data, paving the way for more accessible, efficient, and accurate data analysis in clinical research.

REFERENCES

[1] Wayne Kubick, Stephen J. Ruberg, and Edward Helton. Toward a comprehensive cdisc submission data standard. *Therapeutic Innovation & Regulatory Science*, 2007. (IF: 3).

[2] Gaurav Luthria and Qingbo Wang. Implementing a cloud based method for protected clinical trial data sharing. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, 2019.

[3] G. Gursoy, C. Brannon, and M. Gerstein. Using ethereum blockchain to store and query pharmacogenomics data via smart contracts. *Bio.bioinformatics*, 2019. (IF: 3).

[4] Mohammed Saeed, Nicola De Cao, and Paolo Papotti. Querying large language models with sql. *arXiv-CS.DB*, 2023. (IF: 3).

[5] Xuedi Qin, Chengliang Chai, Yuyu Luo, Nan Tang, and Guoliang Li. Interactively discovering and ranking desired tuples without writing sql queries. *SIGMOD*, 2020. (IF: 3).

[6] Pedro Orvalho, Miguel Terra-Neves, M. Ventura, R. Martins, and Vasco M. Manquinho. Squares : A sql synthesizer using query reverse engineering. *Proc. VLDB Endow.*, 2020. (IF: 4).

[7] Yuanyuan Tian, En Liang Xu, Wei Zhao, Mir Hamid Pirahesh, Sui Jun Tong, Wen Sun, Thomas Kolanko, Md. Shahidul Haque Apu, and Huijuan Peng. Ibm db2 graph: Supporting synergistic and retrofittable graph queries inside ibm db2. *SIGMOD*, 2020. (IF: 3).

[8] Ziije Zhong, Linqing Zhong, Zhaoze Sun, Qingyun Jin, Zengchang Qin, and Xiaofan Zhang. Synthet2c: Generating synthetic data for fine-tuning large language models on the text2cypher task. *arXiv-CS.AI*, 2024.

[9] Xiaoyu Zhang, Fengjing Yin, Guojie Ma, Bin Ge, and Weidong Xiao. M-sql: Multi-task representation learning for single-table text2sql generation. *IEEE Access*, 2020.

[10] Ayush Kumar, Parth Nagarkar, Prabhav Nalhe, and Sanjeev Vijayakumar. Deep learning driven natural languages text to sql query conversion: A survey. *arXiv preprint arXiv:2201.11460*, 2022.

[11] LangChain. Langchain cypher search: Tips & tricks, 2023.

[12] LangChain. Langchain sql database integration, 2023.

[13] Feifei Pan, Mustafa Canim, Michael Glass, Alfio Gliozzo, and James Hendler. End-to-end table question answering via retrieval-augmented generation. *arXiv preprint arXiv:2201.11460*, 2022.

[14] C. S. Krishna. Prompt generate train (pgt): Few-shot domain adaption of retrieval augmented generation models for open book question-answering. *arXiv preprint arXiv:2301.10035*, 2023.

[15] Zhentao Xu, Mark Jerome Cruz, Matthew Guevara, Tie Wang, Manasi Deshpande, Xiaofeng Wang, and Zheng Li. Retrieval-augmented generation with knowledge graphs for customer service question answering. *arXiv preprint arXiv:2401.56789*, 2024.

[16] Linhao Ye, Zhikai Lei, Jianghao Yin, Qin Chen, Jie Zhou, and Liang He. Boosting conversational question answering with fine-grained retrieval-augmentation and self-check. *arXiv preprint arXiv:2401.56790*, 2024.

[17] Leonid Libkin, Juan L. Reutter, and Domagoj Vrgoc. Trial for rdf: Adapting graph query languages for rdf data. 2013. (IF: 4).

APPENDIX: Generated Questions Overview

Categorization of LLM-generated questions based on retrieved protocol info for clinical trial data

A Simple Questions

1. What is the average age of the participants in the study?

2. How many male and female participants are there?

3. What is the distribution of participants by race?

4. How many participants have reported any adverse events?

5. What is the average body mass index (BMI) of the participants?

6. How many participants experienced severe adverse events?

7. What is the most common type of adverse event reported?

8. How many participants are in each treatment group?

9. What is the average height of the participants?

10. How many participants had a prior history of cancer-related surgeries?

11. What is the distribution of participants by ethnicity?

12. How many participants are categorized under each ECOG performance status at baseline?

13. What is the average weight of the participants?

14. How many participants received each type of concomitant medication?

15. How many participants reported at least one serious adverse event?

16. How many participants are below the age of 65?

17. What is the average duration of response for the participants who responded?

18. How many participants have a stable disease as their best overall response?

19. What is the median progression-free survival (PFS) for the participants?

20. How many participants are receiving concomitant medications?

21. What is the average systolic blood pressure of the participants?

22. How many participants have experienced dose-limiting toxicities?

23. What is the average diastolic blood pressure of the participants?

24. How many participants are categorized as having progressive disease?

25. What is the average duration of therapy for the participants?

26. How many participants are of child-bearing potential?

27. What is the average temperature of the participants?

28. How many participants have a complete response as their best overall response?

29. What is the average heart rate of the participants?

30. How many participants discontinued the study due to adverse events?

B Medium Questions

31. What is the average age of the participants in each treatment group?

32. How many participants in each race category have reported adverse events?

33. What is the distribution of adverse events by severity for the participants?

34. How many participants experienced each type of adverse event in the study?

35. What is the average BMI for participants who experienced severe adverse events?

36. How many participants in each treatment group reported serious adverse events?

37. What is the distribution of adverse events by system organ class (SOC)?

38. How many participants experienced adverse events related to study treatment?

39. What is the average weight for participants in each treatment group?

40. How many participants with a history of cancer-related surgeries reported adverse events?

41. What is the distribution of participants by ECOG performance status at baseline for each treatment group?

42. How many participants experienced dose-limiting toxicities in each treatment group?

43. What is the median progression-free survival (PFS) for each treatment group?

44. How many participants in each ethnicity category experienced serious adverse events?

45. What is the average height of participants who reported at least one adverse event?

46. How many participants received concomitant medications in each treatment group?

47. What is the average systolic blood pressure for participants in each treatment group?

48. How many participants with stable disease as their best overall response are in each treatment group?

49. What is the average diastolic blood pressure for participants who reported serious adverse events?

50. How many participants discontinued the study due to adverse events in each treatment group?

51. What is the average temperature of participants in each treatment group?

52. How many participants with a complete response as their best overall response are in each treatment group?

53. What is the average heart rate for participants who experienced dose-limiting toxicities?

54. How many participants reported at least one adverse event in each age category?

55. What is the distribution of participants by race in each treatment group?

56. How many participants with progressive disease are in each treatment group?

57. What is the average duration of therapy for participants in each treatment group?

58. How many participants of child-bearing potential experienced serious adverse events?

59. What is the average BMI for participants who reported serious adverse events?

60. How many participants with a prior history of cancer-related surgeries are in each treatment group?

C Difficult Questions

61. What is the average age of participants who experienced each type of adverse event?

62. How many participants in each race category reported dose-limiting toxicities?

63. What is the distribution of adverse events by severity for participants in each treatment group?

64. How many participants experienced serious adverse events related to study treatment in each treatment group?

65. What is the average BMI for participants who experienced each type of serious adverse event?

66. How many participants in each ethnicity category reported adverse events?

67. What is the distribution of adverse events by system organ class (SOC) for each treatment group?

68. How many participants in each age category experienced dose-limiting toxicities?

69. What is the average weight of participants who experienced dose-limiting toxicities?

70. How many participants with stable disease as their best overall response reported adverse events?

71. What is the median progression-free survival (PFS) for participants in each age category?

72. How many participants in each treatment group reported serious adverse events by system organ class (SOC)?

73. What is the average height of participants who experienced serious adverse events related to study treatment?

74. How many participants in each race category experienced stable disease as their best overall response?

75. What is the average BMI for participants in each treatment group who reported at least one adverse event?

76. How many participants in each treatment group received concomitant medications by type of medication?

77. What is the average systolic blood pressure for participants who experienced each type of adverse event?

78. How many participants in each treatment group experienced serious adverse events by severity?

79. What is the average diastolic blood pressure for participants who reported adverse events by severity?

80. How many participants in each treatment group discontinued the study due to serious adverse events?

81. What is the average temperature for participants who experienced dose-limiting toxicities in each treatment group?

82. How many participants in each treatment group experienced dose-limiting toxicities by severity?

83. What is the average heart rate for participants who reported each type of adverse event?

84. How many participants in each treatment group reported at least one adverse event by system organ class (SOC)?

85. What is the distribution of participants by ethnicity in each treatment group?

86. How many participants in each age category reported serious adverse events by system organ class (SOC)?

87. What is the average duration of therapy for participants who reported at least one serious adverse event?

88. How many participants of child-bearing potential experienced dose-limiting toxicities in each treatment group?

89. What is the average BMI for participants in each treatment group who reported serious adverse events by severity?

90. How many participants with a prior history of cancer-related surgeries reported dose-limiting toxicities? Last edited just now

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name:	Jaime Yan
Company:	Merck & Co., Inc.
Address:	2025 E Scott Ave, Rahway, NJ, USA
Email:	mingyu.yan1@merck.com

Author Name:	Changhong Shi
Company:	Merck & Co., Inc.
Address:	2025 E Scott Ave, Rahway, NJ, USA
Work Phone:	+1 (732) 5941383
Email:	changhong_shi@merck.com