



Enhancing Clinical Trial Data Queries with LLMs and Neo4j

A Flexible Framework for ADaM Dataset Dataset Management

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

Agenda

1

1. Background & Motivation

We'll discuss current challenges in clinical data management and the need for improved query methods.

2

2. Technical Framework

An overview of our system architecture and implementation approach.

3

3. Results & Analysis

We'll present performance metrics and a comparative analysis of our new approach.

4

4. Impact & Future Directions

Exploring clinical trial applications and our development roadmap.

Background & Motivation

Current State

CDISC ADaM standards are an industry requirement, with traditional SQL queries dominating current practice. Excel-based specifications are widely used, but they come with limitations.

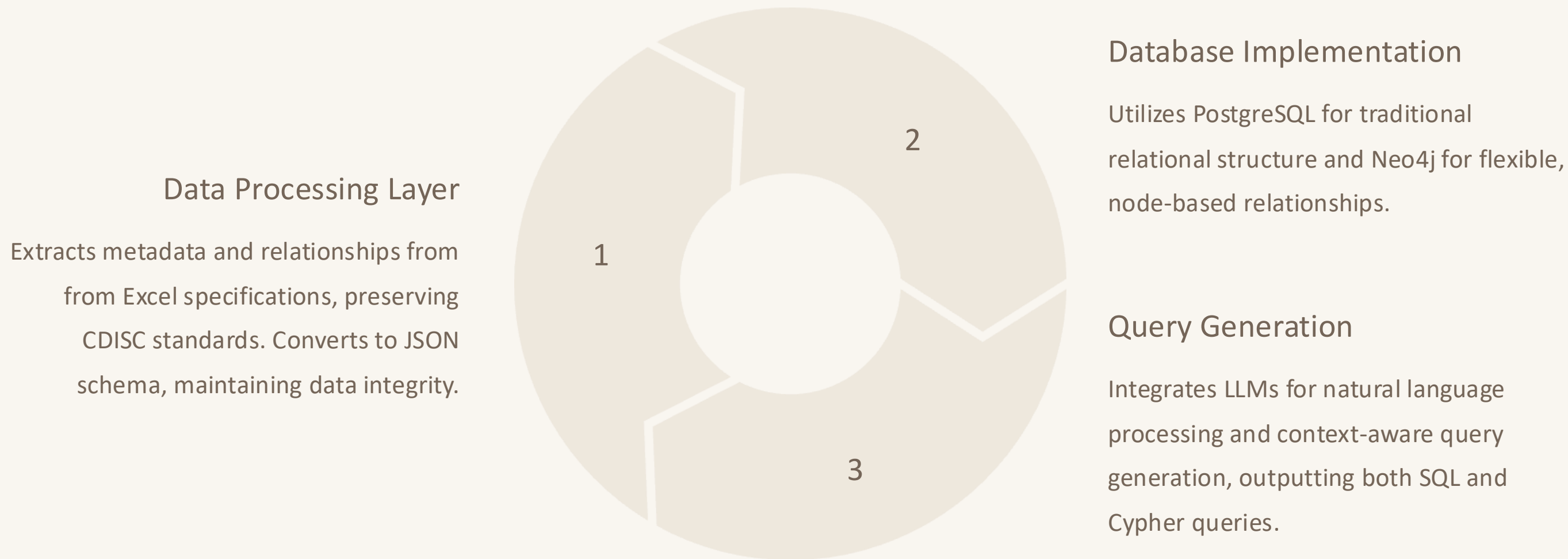
Key Challenges

Complex data relationships are difficult to query, there's limited flexibility in data exploration, and a high level of expertise is required for query writing. The data retrieval process is often time-consuming.

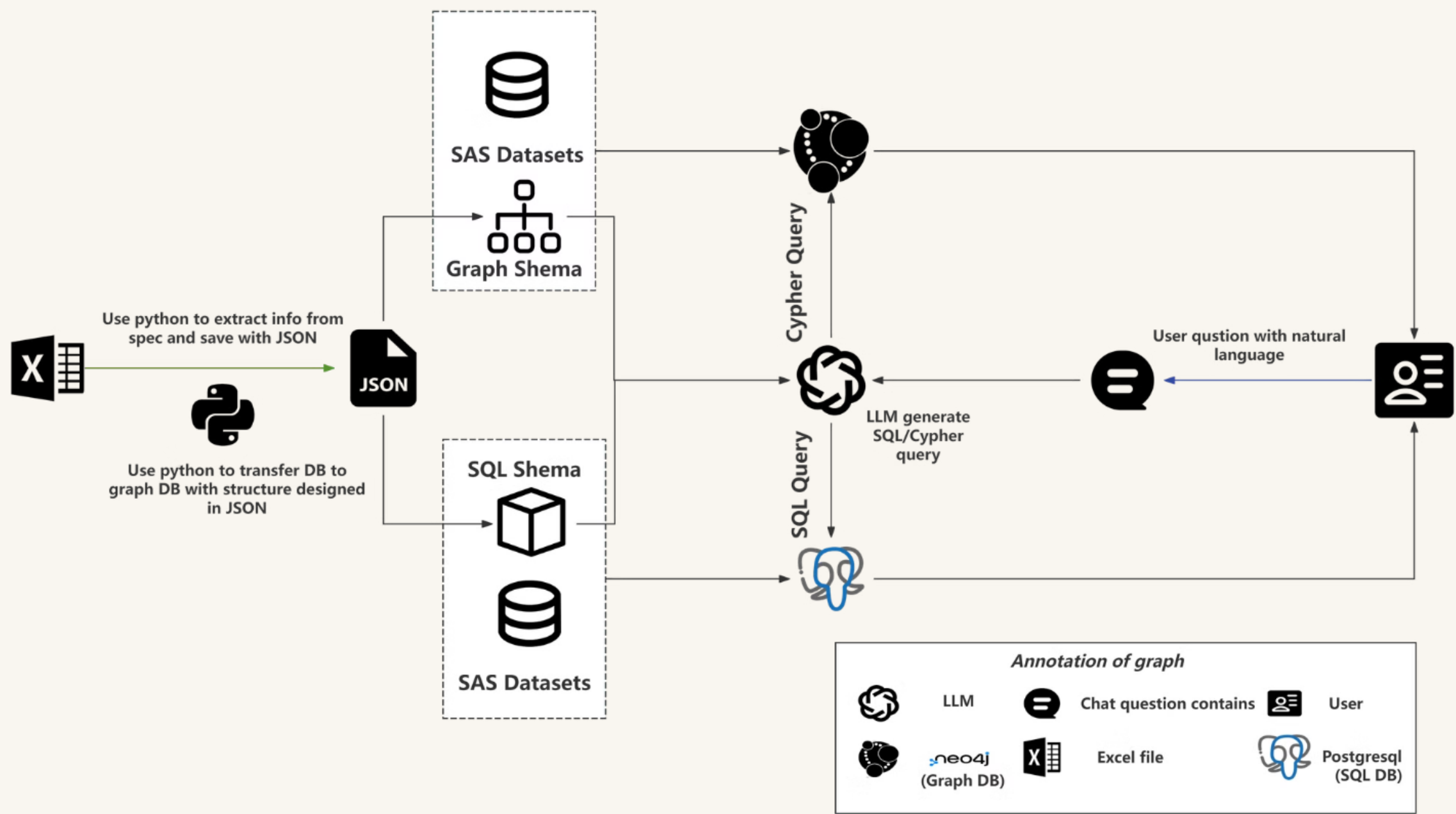
Industry Needs

There's a growing demand for more intuitive query methods, better handling of interconnected data, natural language interfaces, and improved accuracy in complex queries.

Technical Framework Architecture



Framework Flow



Data Model Design

ADSL (Subject-Level)

Contains demographics, baseline characteristics, with one record per subject.

ADAE (Adverse Events)

Used for safety monitoring, with multiple events possible per per subject.

ADCM (Concomitant Medications)

Tracks treatments and temporal relationships between medications.

ADLB/ADVS

Covers laboratory results and vital signs monitoring throughout throughout the trial.

SQL Database Schema Design

Key Design Principles

- **Primary Keys:** USUBJID as primary identifier in ADSL
- **Foreign Keys:** Linking all supplementary datasets to ADSL
- **Data Types:** Appropriate type selection for regulatory compliance
- **Constraints:** Enforcing data integrity and relationships

Python code to generate the SQLschema

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
```

Index Strategy

- Primary key indexes
- Frequently queried columns
- Join optimization indexes
- Performance-critical fields

The SQL schema

SQL Schema (ADSL Table)

```
CREATE TABLE adsl (
    -- Core identifiers
    STUDYID VARCHAR NOT NULL COMMENT 'Study Identifier',
    USUBJID VARCHAR NOT NULL COMMENT 'Unique Subject Identifier',
    SUBJID VARCHAR COMMENT 'Subject Identifier for the Study',
    SITEID VARCHAR COMMENT 'Study Site Identifier',

    -- Demographic information
    AGE INTEGER COMMENT 'Age',
    AGEU VARCHAR COMMENT 'Age Units',
    AGEGR1 VARCHAR COMMENT 'Pooled Age Group 1',
    SEX VARCHAR COMMENT 'Sex',

    PRIMARY KEY (USUBJID)
);
```

Graph Database Schema

Node Types & Properties

- Subject Nodes: Unique identifiers, demographic attributes, study participation details
- Event Nodes: Adverse events, laboratory measurements, vital signs records
- Category Nodes: Classification elements, grouping attributes, reference data

Python code to generate the Cypher schema

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

Relationships

- PARTICIPATED_IN
- EXPERIENCED
- ASSIGNED_TO
- HAS_CATEGORY

Cyther schema

```
Neo4j Schema Definition

// 1. Create Constraints for unique properties
CREATE CONSTRAINT study_id IF NOT EXISTS
FOR (s:Study) REQUIRE s.STUDYID IS UNIQUE;

CREATE CONSTRAINT site_id IF NOT EXISTS
FOR (s:Site) REQUIRE s.SITEID IS UNIQUE;

CREATE CONSTRAINT subject_id IF NOT EXISTS
FOR (s:Subject) REQUIRE s.USUBJID IS UNIQUE;

// 2. Create indexes for performance
CREATE INDEX study_index IF NOT EXISTS
FOR (s:Study) ON (s.STUDYID);

CREATE INDEX site_index IF NOT EXISTS
FOR (s:Site) ON (s.SITEID);

CREATE INDEX subject_index IF NOT EXISTS
FOR (s:Subject) ON (s.USUBJID);

CREATE INDEX age_group_index IF NOT EXISTS
FOR (s:AgeGroup) ON (s.value);

CREATE INDEX sex_category_index IF NOT EXISTS
FOR (s:SexCategory) ON (s.value);

// 3. Node Labels and Properties Definition
(:Study {
  STUDYID: string
})

(:Site {
  SITEID: string
})

(:Subject {
  USUBJID: string
})

(:AgeGroup {
  value: string,
  order: integer
})

(:SexCategory {
  value: string,
  order: integer
})

// 4. Relationship Types
(:Subject)-[:PARTICIPATED_IN]->(:Study)
(:Subject)-[:ASSIGNED_TO]->(:Site)
(:Subject)-[:HAS_AGE_GROUP]->(:AgeGroup)
(:Subject)-[:HAS_SEX]->(:SexCategory)
```

LLM Query Processing

1. Input Processing

Natural language parsing, context extraction, and intent recognition.

Example Transformation:

Natural Language: "Find all patients who experienced severe adverse events"

SQL: `SELECT DISTINCT p.* FROM patients p JOIN adverse_events ae ON p.id = ae.patient_id WHERE ae.severity = 'SEVERE';`

Cypher: `MATCH (p:Patient)-[:EXPERIENCED]->(ae:AdverseEvent) WHERE ae.severity = 'SEVERE' RETURN DISTINCT p;`

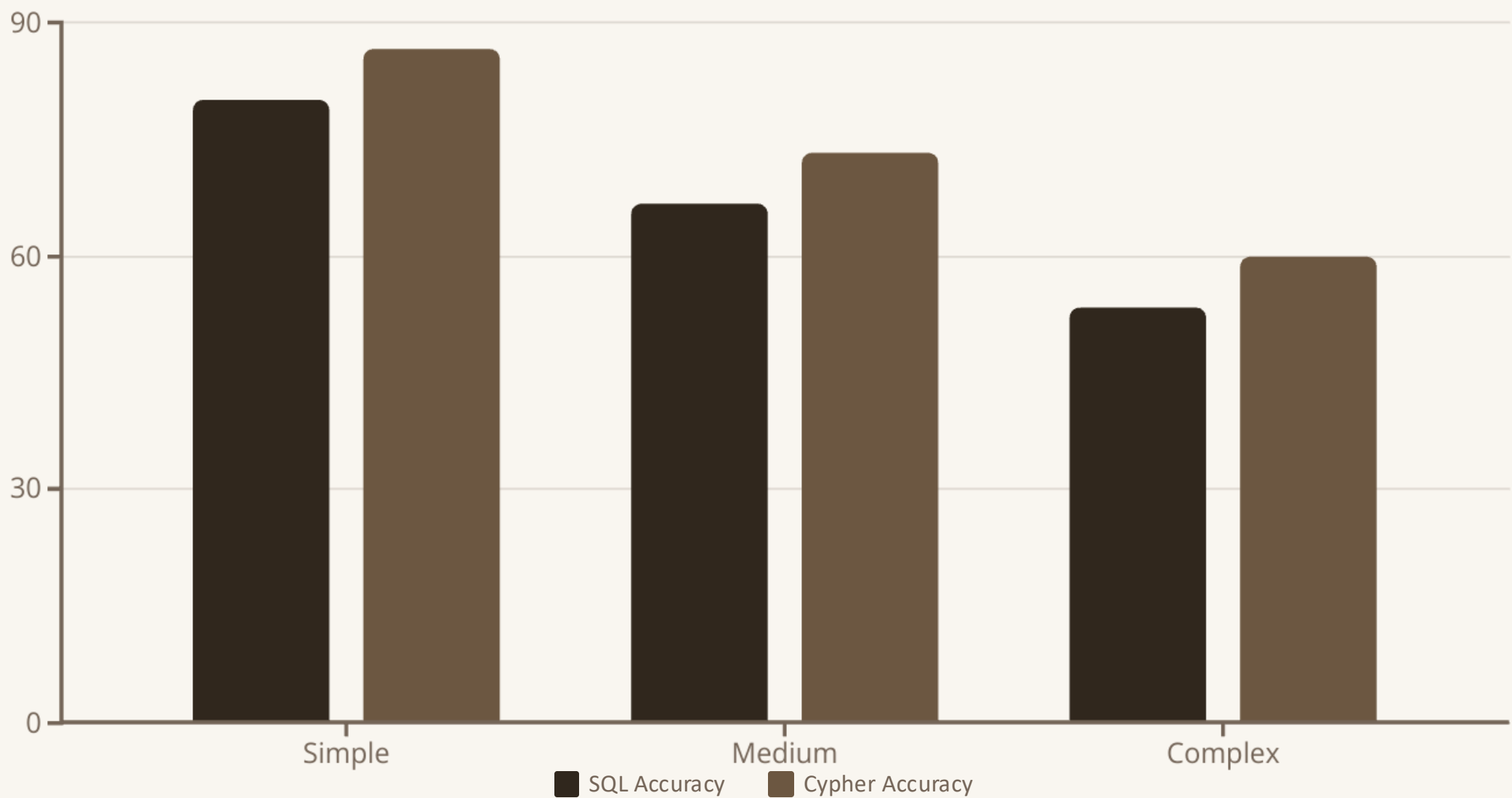
2. Schema Mapping

Entity identification, relationship mapping, and constraint detection.

3. Query Construction

SQL/Cypher syntax generation, parameter binding, and optimization rules.

Performance Analysis



Our comprehensive testing analyzed 90 queries across three complexity levels. Cypher consistently outperformed SQL, with the performance gap increasing for more complex queries. The results show significant advantages in relationship-heavy queries, though lower F1 scores indicate room for improvement in precision.

Implementation Benefits

Clinical Trial Impact

Achieved 73.33% overall accuracy with Cypher, reduced query development time, improved visibility of complex data relationships, and enhanced capabilities for exploratory data analysis.

Operational Advantages

Natural language interface reduces technical barriers, enables faster query development and testing cycles, provides intuitive data relationship modeling, and offers robust support for ad-hoc analysis needs.

Cost Considerations

Initial setup investment for Neo4j implementation is offset by reduced long-term maintenance costs. Training requirements for team transition are balanced by positive ROI through efficiency gains.

Challenges & Solutions

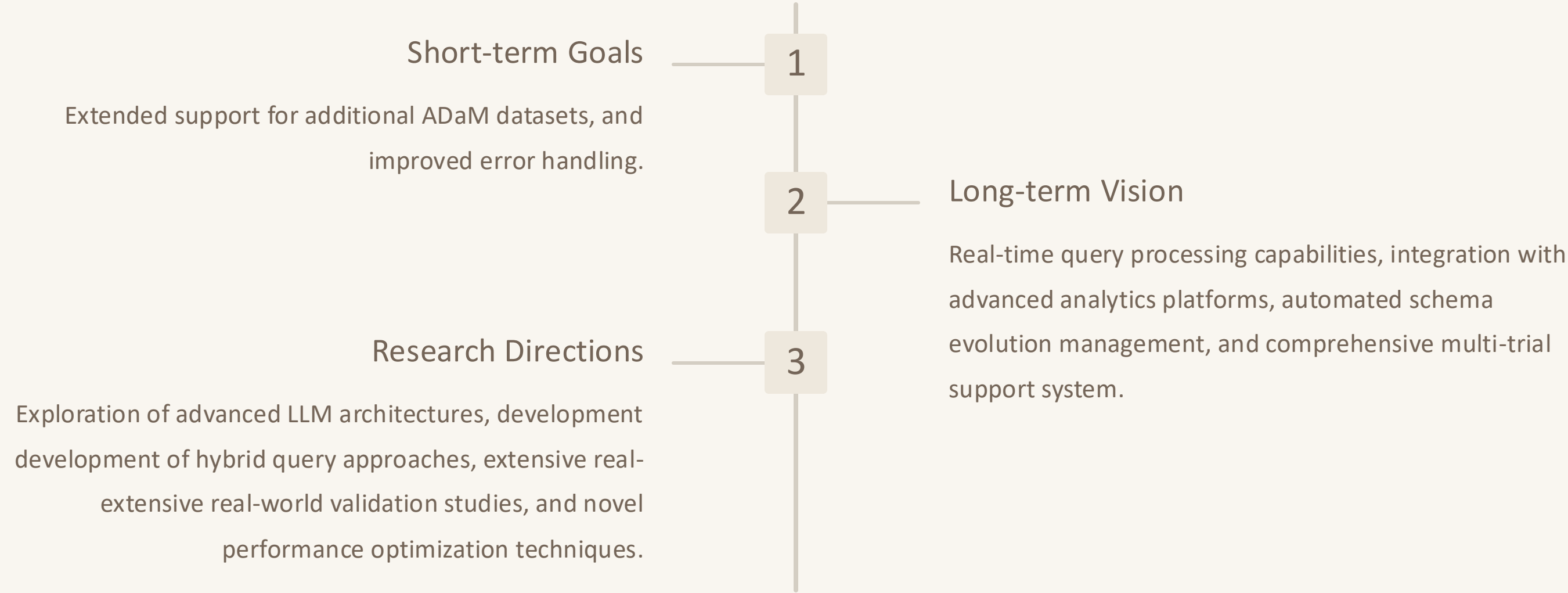
Technical Challenges

- Schema Conversion: Solved with automated mapping tools and validation frameworks
- Query Accuracy: Addressed through LLM fine-tuning and continuous learning
- Performance Optimization: Implemented advanced indexing and caching mechanisms

Integration Challenges

- Legacy Systems: Managed with phased implementation and parallel running
- User Adoption: Supported by comprehensive training and documentation

Future Development



Conclusion

1 Key Achievements

Successfully integrated LLMs with with graph database, demonstrated demonstrated improved query accuracy, enhanced data accessibility accessibility for users, and established established a framework for future future development.

2 Value Proposition

More efficient clinical trial data analysis, significantly reduced technical barriers, enhanced decision decision support capabilities, and and improved regulatory compliance compliance measures.

3 Next Steps

Conduct comprehensive industry validation, implement feature enhancements, optimize performance performance metrics, and support support wider user adoption.

Questions & Contact

Jaime Yan

mingyu.yan1@merck.com

Changhong Shi

changhong_shi@merck.com

We welcome your questions and feedback. Please feel free to reach out to us for further discussion on our framework for enhancing clinical trial enhancing clinical trial data queries.