BIOGRAPHX

1. Executive Summary

Project Title
Graph-Augmented Agentic AI for Explainable Biomedical Question Answering

One-liner
Build a biomedical question answering (QA) system that doesn't just answer, but **shows its work** using:

- A **Knowledge Graph (Neo4j)**
- A **Vector Store (ChromaDB)** over PubMed-like abstracts
- A **fine-tuned open-source HuggingFace QA model**
- A **multi-agent pipeline (LangGraph)** coordinating 6 agents

Data comes entirely from **Kaggle**.
Models are **open-source** and **fine-tuned by you** (no external LLM APIs).

---

2. Objectives & Success Criteria

2.1 Main Objective

Create an **Explainable Biomedical QA system** that:

- Answers natural language medical questions
- Uses **graph reasoning** plus **retrieval-augmented generation (RAG)**

- Provides **evidence sentences and graph paths** supporting the answer

2.2 Success Criteria

The project is successful if:

- ✅ System answers questions end-to-end via a **Streamlit UI**
- ✅ Each answer includes **at least one evidence sentence with PMID**
- ✅ A **Neo4j subgraph** (e.g., Drug → TREATS → Disease) is shown for relevant questions
- ✅ QA model achieves reasonable metrics (e.g., EM/F1) on a subset of **PubMedQA**
- ✅ Code is reproducible and documented well enough for another student to run

---

3. Problem & Business Understanding

3.1 Motivation

Biomedical professionals and students face:

- Huge volume of literature (PubMed, clinical studies)
- Fragmented knowledge about diseases, drugs, symptoms
- Black-box AI models that don't explain why they answered something

In medicine, **wrong but confident answers are dangerous**. We need systems that **both answer and explain.**

3.2 Business / Real-World Use

Potential use cases:

- Clinical decision support (junior doctors check treatments)
- Research assistants (find relationships between drugs and diseases)
- Educational tools (explain reasoning paths to students)

The project is a **prototype** for such systems, focusing on transparency, graphs, and agents.

---

4. Data Understanding (Kaggle Only)

You will use **three main Kaggle datasets**:

4.1 MedQuAD – Medical Question Answer Dataset

- Q&A pairs from NIH websites
- Used for:
    - Training/fine-tuning QA model
    - Building graph nodes (disease, drug, symptom, etc.)

4.2 PubMed Abstracts Dataset

- Large collection of PubMed-like abstracts
- Used as:
    - Evidence corpus for retrieval
    - Source of sentences to support final answers

4.3 PubMedQA

- Biomedical QA benchmark
- Used for **evaluation only**
- Questions + associated abstracts + gold answers

4.4 ETL Overview

ETL (Extract → Transform → Load) converts raw Kaggle datasets into **clean, structured, machine-ready files** stored in data/processed/.

ETL Inputs (raw data from Kaggle)

Stored in:

data/raw/

```
│
├── medquad.csv
├── pubmed_abstracts.csv
├── pubmed_qa_pga_labeled.parquet
└── pubmed_qa_pga_artificial.parquet
```

ETL Outputs

Stored in:

data/processed/

```
│
├── medquad_clean.csv
├── pubmedqa_clean.csv
├── pubmed_sentences.parquet
├── entity_mappings.csv
└── graph/
    ├── nodes_disease.csv
    ├── nodes_drug.csv
    ├── nodes_question.csv
    └── rels_about.csv
```

ETL Pipeline Steps

1) clean_medquad.py

Input: medquad.csv
Output: medquad_clean.csv
Purpose: Clean Q/A text → structure → remove noise

2) clean_pubmedqa.py

Input: PubMedQA parquet files
Output: pubmedqa_clean.csv
Purpose: Normalize PubMedQA evaluation dataset

3) prepare_pubmed_sentences.py

Input: PubMed abstracts
Output: pubmed_sentences.parquet
Purpose: Split abstracts → sentences → store efficiently for ChromaDB

4) extract_entities.py

Input: medquad_clean.csv
Output: entity_mappings.csv
Purpose: SciSpaCy biomedical NER → extract Disease/Drug/Symptom

5) build_graph_csvs.py

Input: entity_mappings + MedQuAD
Output: Neo4j node & relationship CSVs
Purpose: Build graph ingestion files

---

5. Tools & Technologies

5.1 Data & Core Libraries

- pandas, numpy, scipy, tqdm
- matplotlib, seaborn (for simple plots & EDA)

5.2 NLP & Biomedical NER

- transformers, sentence-transformers (HuggingFace)
- datasets (HuggingFace Datasets)
- spacy, scispacy (for biomedical NER)

5.3 Graph Tools

- Neo4j (Docker or Desktop)
- py2neo (Python driver)
- networkx (for graph manipulation/visualization)

5.4 Vector Store & Retrieval

- chromadb
- faiss-cpu (embedded within/under Chroma)

5.5 Agentic Framework / RAG Orchestration

- langchain
- langgraph

5.6 Model Fine-Tuning

- torch, accelerate
- peft (for LoRA)
- bitsandbytes (optional for 8-bit/4-bit finetuning)

5.7 UI & Visualization

- streamlit
- pyvis or networkx for graph visualization

5.8 Testing & Evaluation

- pytest, pytest-cov
- evaluate, rouge-score

---

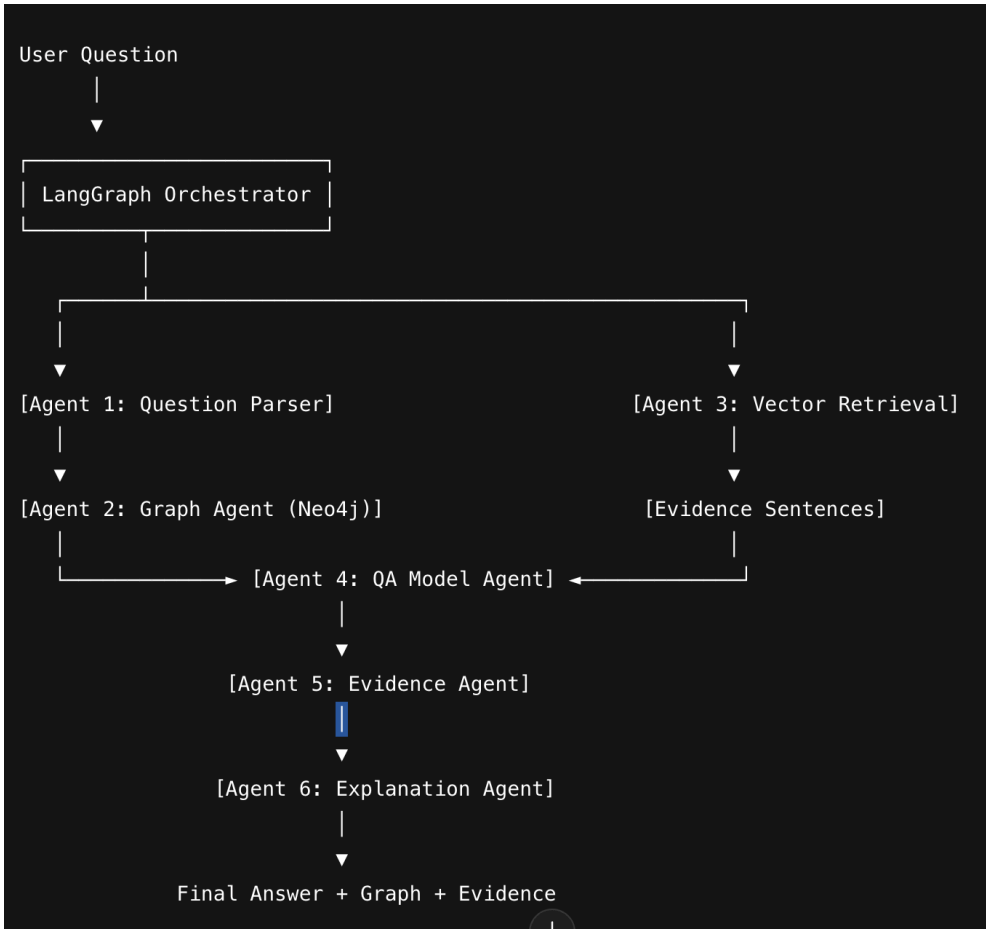
6. System Architecture (High-Level)

6.1 Big Picture

1. User asks a **biomedical question** in Streamlit.
2. **LangGraph** orchestrates 6 agents:
    - Question Parsing Agent
    - Graph Query Agent (Neo4j)
    - Vector Retrieval Agent (ChromaDB)
    - QA Model Agent (fine-tuned HF model)
    - Evidence Validation Agent
    - Explanation Agent
3. The system returns:
    - Answer
    - Evidence sentences (with PMIDs)

- Graph path(s) showing relationships

6.2 Architecture Diagram (Logical)



---

7. Agentic Workflow (6 Agents – Beginners Can Understand)

Agent 1 – Question Parsing & Classification

- Input: user's question string

- Responsibilities:
    - Detect question type (e.g., “treatment”, “symptoms”, “cause”)
    - Extract biomedical entities (disease names, drugs, etc.) using SciSpaCy
- Output:
    - {"question_type": "...", "entities": ["..."]}

Agent 2 – Graph Query Agent (Neo4j + Cypher)

- Input: entities + question_type
- Responsibilities:
    - Generate Cypher queries to explore knowledge graph
    - Fetch related nodes and edges for those entities
    - Summarize graph info (e.g., “Metformin TREATS Type 2 Diabetes”)
- Output:
    - List of triples or edges like (node1, relation, node2)

Agent 3 – Vector Retrieval Agent (ChromaDB)

- Input: question + (optional) entities
- Responsibilities:
    - Encode question → embedding (sentence-transformers)
    - Retrieve top-k similar PubMed sentences
    - Attach PMIDs and metadata
- Output:
    - [{"pmid": ..., "sentence": "...", "score": ...}, ...]

Agent 4 – QA Model Agent (Fine-tuned HF Model)

- Input: question + graph summary + top evidence sentences
- Responsibilities:
    - Construct a prompt/input (e.g. for FLAN-T5)
    - Generate an answer sequence
- Output:
    - Answer text (e.g., “Metformin and insulin are common treatments for type 2 diabetes.”)

Agent 5 – Evidence Validation Agent

- Input: answer + evidence sentences
- Responsibilities:
    - Check which sentences strongly support the answer
    - Optionally, use a simple similarity scoring between answer and each sentence
- Output:
    - Ranked evidence; top-N sentences retained

Agent 6 – Explanation & Output Agent

- Input: answer + graph triples + evidence list
- Responsibilities:
    - Build structured final output:
        - Explanation text if needed
        - Evidence list with PMIDs
        - Graph paths (e.g., drug–disease–symptom)
- Output (JSON-like):

{

"answer": "Metformin is commonly used to treat type 2 diabetes.",

```
  "graph_path": [
    ["Metformin", "TREATS", "Type 2 Diabetes"]
  ],
  "evidence": [
    {"pmid": "12345", "sentence": "Metformin is a first-line therapy ..."}
  ]
}
```

---

8. Repository Structure (Project Layout)

```
BIOGRAPHX/
│
├── README.md
├── requirements.txt
├── .gitignore
│
├── data/
│   ├── raw/
│   │   ├── medquad.csv
│   │   ├── pubmed_abstracts.csv
│   │   ├── pubmed_qa_pga_labeled.parquet
```

```
│   │   └── pubmed_qa_pga_artificial.parquet
│   │
│   └── processed/
│       ├── medquad_clean.csv
│       ├── pubmedqa_clean.csv
│       ├── pubmed_sentences.parquet
│       ├── entity_mappings.csv
│       └── graph/
│           ├── nodes_disease.csv
│           ├── nodes_drug.csv
│           ├── nodes_symptom.csv  (optional)
│           ├── nodes_question.csv
│           └── rels_about.csv
│
├── etl/
│   ├── clean_medquad.py
│   ├── clean_pubmedqa.py
│   ├── prepare_pubmed_sentences.py
│   ├── extract_entities.py
│   ├── build_graph_csvs.py
│   └── utils/
```

```
│       ├── __init__.py
│       └── text_cleaning.py
│
├── graph/
│   ├── schema.cql
│   ├── load_graph.py
│   ├── extract_entities.py   (if used separately)
│   └── queries/
│       ├── get_disease_relations.cql
│       ├── get_drug_relations.cql
│       └── get_question_entities.cql
│
├── rag/
│   ├── prepare_pubmed_sentences.py    (duplicate optional)
│   ├── build_index.py
│   ├── retriever.py
│   └── utils/
│       └── embedding_models.py
│
├── agents/
│   ├── question_agent.py
```

```
│   ├── graph_agent.py
│   ├── retriever_agent.py
│   ├── qa_model_agent.py
│   ├── evidence_agent.py
│   ├── explanation_agent.py
│   └── agent_graph.py
│
├── training/
│   ├── prepare_medquad_dataset.py
│   ├── finetune_qa_model.py
│   ├── test_finetuned_model.py
│   ├── eval_pubmedqa.py
│   └── utils/
│       └── tokenizer_utils.py
│
├── models/
│   └── fine_tuned/
│       └── flan_t5_medquad/
│           ├── config.json
│           ├── adapter_model.bin
│           ├── tokenizer.json
```

```
│           └── special_tokens_map.json
│
├── app/
│   ├── streamlit_app.py
│   └── components/
│       ├── graph_viz.py
│       └── evidence_panel.py
│
├── configs/
│   ├── config.yaml
│   ├── neo4j.yaml
│   ├── chroma.yaml
│   └── training.yaml
│
├── notebooks/
│   ├── 01_eda_medquad.ipynb
│   ├── 02_pubmed_analysis.ipynb
│   ├── 03_error_analysis.ipynb
│   └── 04_agent_pipeline_debug.ipynb
│
├── tests/
```

```
│   ├── test_data_loading.py
│   ├── test_graph_build.py
│   ├── test_retriever.py
│   ├── test_agents.py
│   └── test_pipeline_integration.py
│
└── scripts/
    ├── run_neo4j_docker.sh
    ├── build_all_etl.sh
    ├── run_all_tests.sh
    └── demo_pipeline.py
```

---

9. Environment & Setup (Beginner-Friendly)

9.1 Create Project & Virtual Environment

mkdir graph-agentic-biomed-qa

cd graph-agentic-biomed-qa

python -m venv .venv

Activate

Windows:

.venv\Scripts\activate

macOS/Linux:

source .venv/bin/activate

9.2 Create requirements.txt and Install

Use the dependency list you already have (with datasets, peft, bitsandbytes, etc.) and run:

pip install -r requirements.txt

9.3 Initialize Git

git init

echo ".venv/" >> .gitignore

echo "__pycache__/" >> .gitignore

echo "data/raw/" >> .gitignore

Project Implementation Plan

🟦 Sprint 1 – Project Bootstrapping & Data Download (DETAILED)

Goal:

Get the project repository, environment, dependencies, and raw datasets ready with basic exploration.

Tasks:

1. Create Repository & Virtual Environment

- mkdir graph-agentic-biomed-qa
- Initialize Git repo
- Create virtual environment (python -m venv .venv)
- Activate venv
- Add .venv/, __pycache__/, and data/raw/ to .gitignore

2. Install Requirements

- Create requirements.txt containing:
    - pandas, numpy, tqdm
    - scispacy, spacy
    - transformers, sentence-transformers
    - chromadb, faiss-cpu
    - py2neo, neo4j

    - torch, accelerate, peft
    - streamlit, pyvis
- Install:
  pip install -r requirements.txt

3. Download Kaggle Datasets

Use Kaggle CLI or manual download:

- **MedQuAD** → data/raw/medquad/
- **PubMed Abstracts** → data/raw/pubmed/
- **PubMedQA** → data/raw/pubmedqa/

4. Create EDA Notebook

File: notebooks/01_eda_medquad.ipynb

Perform initial understanding:

- Load MedQuAD JSON / CSV
- Show .head()
- Print a couple Q/A entries
- Count rows
- Identify missing values
- Basic text length statistics

Deliverables:

- Working virtual environment
- Raw Kaggle datasets inside data/raw/
- EDA notebook demonstrating dataset familiarity

🟩 Sprint 2 – Data Cleaning, NER & Knowledge Graph Schema (FULL DETAILED VERSION)

(All content below matches your formatting but expanded)

Goal:

Clean MedQuAD, extract biomedical entities using SciSpaCy, and design the Neo4j graph schema.

Tasks:

---

1. Clean MedQuAD – training/prepare_medquad_dataset.py

Actions:

- Load raw medquad.csv
- Normalize text:
    - trim whitespace
    - remove HTML tags
    - unify punctuation
    - lowercase (optional)

- Keep only relevant columns:
    - question
    - answer
    - source_url *(optional)*
- Drop duplicates and rows where Q/A is missing
- Save cleaned dataset to:
  data/processed/medquad_clean.csv

Purpose:

This dataset becomes:

- Input to NER
- Graph Question nodes
- Training set for QA model fine-tuning

---

2. Install SciSpaCy + Perform Biomedical NER – graph/extract_entities.py

Actions:

Install SciSpaCy model:

pip install scispacy

pip install https://.../en_ner_bc5cdr_md-0.5.0.tar.gz

-
- For each question + answer:
    - run SciSpaCy

    - extract named entities

- Map entities to:
    - **Disease**
    - **Drug**
    - **Symptom**
    - **Chemical** *(optional)*
- Store:
    - entity_text
    - entity_type
    - originating_question
- Save output as:
  data/processed/entity_mappings.csv

Purpose:

Provides **all entity nodes** for the knowledge graph.

---

3. Design Graph Schema – graph/schema.cql

Actions:

Define Neo4j labels:

- :Question
- :Disease

- :Drug
- :Symptom

Define relationships:

- (:Question)-[:ABOUT]->(:Disease | :Drug | :Symptom)
- (Optional) :TREATS, :CAUSES, :HAS_SYMPTOM

Define constraints:

```
CREATE CONSTRAINT question_unique IF NOT EXISTS
ON (q:Question) ASSERT q.id IS UNIQUE;

CREATE CONSTRAINT disease_unique IF NOT EXISTS
ON (d:Disease) ASSERT d.name IS UNIQUE;
```

Purpose:

Creates the structure for the BioGraphX Neo4j Knowledge Graph.

---

4. Build Graph CSVs – etl/build_graph_csvs.py

Actions:

- Convert entity mappings into node CSVs:
    - nodes_disease.csv
    - nodes_drug.csv

    - nodes_symptom.csv
    - nodes_question.csv
- Create ABOUT relationships CSV:
    - rels_about.csv linking Q → entities
- Save all to:
  data/processed/graph/

Purpose:

These files are consumed by Neo4j loader in Sprint 3.

Deliverables:

- medquad_clean.csv
- entity_mappings.csv
- graph/schema.cql
- Full graph CSV files (nodes + edges)
- Graph schema documentation in README

🟨 Sprint 3 – Neo4j Graph Build + PubMed Sentence Embeddings + Chroma Setup (FULL DETAILED VERSION)

Goal:

Load graph into Neo4j, prepare PubMed sentence corpus, embed sentences using BioBERT, and build Chroma vector store.

Tasks:

---

1. Start Neo4j DB

Choose one:

Option A — Docker

```
docker run \
  -p7474:7474 -p7687:7687 \
  -d --name neo4j \
  -e NEO4J_AUTH=neo4j/password \
  neo4j:5.2
```

Option B — Neo4j Desktop

- Create project
- Enable Bolt
- Enable APOC if needed

---

2. Load Graph – graph/load_graph.py

Actions:

- Connect using py2neo
- Apply schema constraints from schema.cql
- Load node CSVs:

    - Disease, Drug, Symptom, Question

- Load relationship CSV:
    - ABOUT edges

Purpose:

Builds the **Biomedical Knowledge Graph** used by Agent 2.

---

3. Prepare PubMed Sentence Corpus – rag/prepare_pubmed_sentences.py

Actions:

- Read pubmed_abstracts.csv
- For each abstract:
    - extract pmid
    - split abstract into individual sentences
- Store rows:

pmid, sentence

- Save to:
  data/processed/pubmed_sentences.parquet

Purpose:

Provides a retrieval-friendly dataset for embedding.

---

4. Build Chroma Vector Index – rag/build_index.py

Actions:

- Load PubMed sentences
- Embed using BioBERT / SciBERT:

```
model = SentenceTransformer("pritamdeka/S-BioBERT-snli-mnli-stsb")
```

- Insert embeddings + metadata into Chroma collection:
  pubmed_sentences
- Persist vector store to disk

Purpose:

Used by Vector Retrieval Agent (Agent 3) to fetch evidence.

---

5. Implement Retrieval – rag/retriever.py

Function:

```
retrieve_sentences(query, k=5)
```

Steps:

- Embed query sentence
- Query Chroma for top-k nearest sentences
- Return:

```
[
```

```
  {"pmid": "...", "sentence": "...", "score": ...},
  ...
]
```

Purpose:

Provides evidence sentences for QA reasoning.

Deliverables:

- Fully populated **Neo4j DB**
- pubmed_sentences.parquet
- Chroma collection pubmed_sentences
- Working retriever returning biomedical evidence

🟧 Sprint 4 – Fine-Tuning HuggingFace QA Model (DETAILED)

Goal:

Train a domain-specific QA model on MedQuAD.

Tasks:

1. Prepare HF Dataset – training/prepare_medquad_dataset.py

- Load medquad_clean.csv
- Build dataset:

{"input": question, "output": answer}

- Split: train/val

---

2. Choose Model

- google/flan-t5-base (recommended)
- Alternatives: t5-small, t5-base

---

3. Write Training Script – training/finetune_qa_model.py

- Tokenize inputs (max_source_length=512)
- Add LoRA configuration for speed
- Train for 2–4 epochs
- Save model to:
  models/fine_tuned/flan_t5_medquad/

---

4. Test Fine-Tuned Model – training/test_finetuned_model.py

- Load saved model
- Run inference on 2–3 sample questions
- Print outputs

Deliverables:

- Fine-tuned model
- Training logs
- Test script confirming basic Q/A performance

🟥 Sprint 5 – Implement the 6-Agent LangGraph Pipeline (DETAILED)

Goal:

Enable the full reasoning workflow using 6 specialized agents.

Tasks:

1. Implement Each Agent (agents/*.py)

question_agent.py

- Input: user question text
- Output:

```
{
 "entities": [...],
 "question_type": ...
}
```

graph_agent.py

- Runs Cypher queries on Neo4j
- Retrieves graph relationships for found entities

retriever_agent.py

- Queries Chroma vector DB
- Returns evidence sentences with PMIDs

qa_model_agent.py

- Combines:
    - question
    - evidence
    - graph context
- Calls fine-tuned T5 model

evidence_agent.py

- Scores sentences vs. model answer
- Picks top explanatory evidence

explanation_agent.py

- Final JSON output:

answer

graph_path

evidence

---

2. Build LangGraph Orchestration – agent_graph.py

- Define node functions
- Define state dictionary
- Connect agents in order:

Question → QuestionAgent → GraphAgent →

RetrieverAgent → QA Agent → EvidenceAgent → ExplanationAgent

---

3. Demo Script – scripts/run_demo_question.py

- Hardcode sample question
- Run pipeline
- Print structured output

Deliverables:

- Six working agent modules
- LangGraph orchestrator
- Working end-to-end demo

🟪 Sprint 6 – Streamlit UI + Evaluation + Final Polish (DETAILED)

Goal:

Create user-friendly frontend, run evaluation, finalize documentation.

Tasks:

---

1. Streamlit UI – app/streamlit_app.py

UI Components:

- Input textbox for biomedical question
- "Ask" button
- Output panels:
    - Final Answer
    - Evidence sentences
    - Graph visualization (pyvis)

---

2. Evaluation – training/eval_pubmedqa.py

Steps:

- Load small PubMedQA sample
- For each question:
    - run your pipeline

    - compare output vs. gold answer
- Compute:
    - Exact Match (EM)
    - F1 Score

---

3. Error Analysis – notebooks/02_error_analysis.ipynb

Inspect failed cases:

- Was evidence relevant?
- Did graph reasoning help?
- Did model hallucinate?
- Are entities missing?

---

4. Final Polish

Update README:

- Setup instructions
- How to run Neo4j
- How to run Streamlit app
- How to run retriever
- How to evaluate

Deliverables:

- Fully working Streamlit app
- Evaluation metrics
- Visual graph explanations
- Error analysis notebook
- Final documentation