QuantEdge Project Timeline (Advanced, with Detailed Vectorization Enhancements)

MIT Data Science Final Year Project

June 29, 2025

1 Overview

QuantEdge is an advanced AI-powered stock trading assistant developed for the MIT Data Science Final Year Project (FYP). It utilizes a hybrid Retrieval-Augmented Generation (RAG) pipeline combining BM25 for keyword search and FAISS for vector search, powered by a fine-tuned DistilBERT model (switchable to Llama-3 5B or Mixtral-7B via a modular interface). The system incorporates an Adaptive Knowledge Graph (AKG) for relational reasoning, Personalized Strategy Synthesis (PSS) for tailored trading strategies, Explainable AI (XAI) for transparent decision-making, a Continuous Learning Ecosystem (CLE) for self-improvement, and an Ethical Compliance Framework (ECF) for data privacy and API compliance. Data from yfinance, NewsAPI, FRED API, and X Platform API is stored in MongoDB Atlas M0, supporting stock prices, news, macro indicators, sentiment, chat history, user preferences, and simulated portfolios. Development occurs in Jupyter Notebook, with Google Colab for GPU-intensive tasks, and a Jupyter-based dashboard enables rich user interactions. Enhanced vectorization optimizes data consumption for RAG and LLM, addressing MongoDB M0s 512MB limit and API constraints through dimensionality reduction, incremental updates, lightweight models, batch processing, and hybrid search tuning.

2 Assumptions

- **Daily Commitment**: 1–2 hours/day, approximately 10 hours/week (5 days/week).
- **Skill Level**: Intermediate proficiency in Python, machine learning, graph databases, and Git.
- **Resources**: Free-tier tools (Jupyter Notebook, Google Colab with T4 GPU, MongoDB Atlas M0, GitHub, yfinance, NewsAPI, FRED API, X Platform API).
- **Start Date**: July 1, 2025.
- **Duration**: 30 weeks (approximately 300 hours).
- **Jupyter Setup**: Local machine with 16GB RAM, optional GPU. Colab provides T4 GPU (16GB VRAM).
- **MongoDB M0**: 512MB storage limit requires efficient indexing, data pruning (>30 days), and JSON archiving.
- **LLM**: DistilBERT initially, with modular interface for switching to Llama-3 or Mixtral.

3 Unique Features

- **Adaptive Knowledge Graph (AKG)**: Models relationships between stocks, sectors, macro indicators, and sentiment in MongoDB, enabling nuanced LLM reasoning via graph-based queries.
- **Personalized Strategy Synthesis (PSS)**: Dynamically generates custom trading strategies by blending user preferences with real-time data from RAG and AKG.
- **Explainable AI (XAI)**: Uses SHAP and AKG to provide transparent, natural language

explanations for trading recommendations.

- **Multi-Modal Conversational Interface**: Integrates text, Plotly charts, and PDF/CSV exports in a Jupyter-based dashboard for user interaction.
- **Continuous Learning Ecosystem (CLE)**: Automates data ingestion, model fine-tuning, and switching for continuous improvement.
- **Ethical Compliance Framework (ECF)**: Ensures compliance with API limits, data privacy, and ethical AI practices through logging and anonymization.
- **Enhanced Vectorization**: Optimizes RAG and LLM performance with dimensionality reduction (PCA/UMAP), incremental embedding updates, lightweight models (`all-MiniLM-L6-v2`), batch processing, and hybrid BM25/FAISS search tuning.

4 Phase 1: Setup and Research (Weeks 1–4, July 1–28, 2025)

Goal: Configure development environments (Jupyter/Colab), set up MongoDB with AKG, initialize Git/GitHub, and research unique features, including vectorization enhancements for RAG and LLM.
Total Hours: 40 hours (10 hours/week).

4.1 Week 1: Environment Setup

- **Set up Jupyter and Colab (3 hours)**:
  - **Objective**: Establish robust development environments for local and cloud-based tasks.
  - **Actions**:
    - Install Jupyter via `pip install jupyter` or Anaconda on a local machine (16GB RAM, Python 3.7+).
    - Create a Google Colab notebook with Python 3.7 and T4 GPU (16GB VRAM) for GPU-intensive tasks.
    - Set up a conda environment (`quantedge-env`) with Python 3.7, ensuring compatibility with libraries.
    - Test Jupyter notebook execution and Colab GPU connectivity with a sample script (e.g., matrix multiplication).
  - **Outcome**: Functional local and cloud environments ready for development and GPU tasks.
  - **Git**: Commit `environment.yml` and `requirements.txt` to `feature/setup` with message "Initial Jupyter/Colab setup".
- **Install dependencies (3 hours)**:
  - **Objective**: Install all required libraries for AKG, PSS, XAI, and enhanced vectorization.
  - **Actions**:
    - Run `pip install pymongo dnspython transformers peft sentence-transformers all-mpnet-base-v2 all-MiniLM-L6-v2 faiss-cpu yfinance newsapi-python fredapi tweepy backtrader matplotlib plotly pdfkit pytest flake8 mlflow shap nbdime schedule networkx scikit-learn umap-learn`.
    - Verify library versions in `requirements.txt` to ensure compatibility (e.g., `transformers==4.30.0`).
    - Test imports in a Jupyter notebook to confirm successful installation.

- **Outcome**: All dependencies installed and verified for development.
- **Git**: Update `requirements.txt`, commit to `feature/setup` with message "Install dependencies for QuantEdge".

- **Initialize Git/GitHub (2 hours)**:
  - **Objective**: Set up version control for collaborative development.
  - **Actions**:
    - Run `git init` in project directory, create a public GitHub repository named `QuantEdge`.
    - Create `README.md` with project overview and `.gitignore` to exclude logs, model checkpoints, and sensitive data.
    - Configure `nbdime` for Jupyter notebook diffs (`nbdime config-git -enable`).
    - Push initial commit to `main` branch.
  - **Outcome**: GitHub repository initialized with version control configured.
  - **Git**: Push to `main`, create `feature/setup` branch, commit with message "Initialize Git/GitHub repository".
- **Configure GitHub Actions (2 hours)**:
  - **Objective**: Automate continuous integration for code quality and testing.
  - **Actions**:
    - Create `.github/workflows/ci.yml` to run `pytest` for unit tests, `flake8` for linting, and notebook execution on push to `feature/*` and `main`.
    - Test CI pipeline with a dummy script to ensure successful execution.
    - Create a GitHub issue to track CI setup progress.
  - **Outcome**: Functional CI pipeline for automated testing and linting.
  - **Git**: Commit CI config to `feature/setup` with message "Configure GitHub Actions CI", create issue.

4.2 Week 2: MongoDB and AKG Setup

- **Set up MongoDB Atlas (4 hours)**:
  - **Objective**: Create a MongoDB Atlas M0 cluster with collections to support AKG and data storage.
  - **Actions**:
    - Create an M0 cluster named `QuantEdge` in MongoDB Atlas (free tier, 512MB).
    - Set up an `atlasAdmin` user with read/write permissions.
    - Configure network access with a static IP for Jupyter (`your.ip/32`) and `0.0.0.0/0` for Colab.
    - Define collections: `stocks`, `news`, `xposts`, `macro_data`, `users`, `strategies`, `strategy_templates`, `embeddings`, `data_versions`, `knowledge_graph`.
    - Create initial schemas in `docs/db-schema.md` (e.g., `stocks`: `{ticker, price, RSI, MACD}`).
  - **Outcome**: MongoDB Atlas M0 cluster operational with defined collections.

- **Git**: Commit schemas to `docs/db-schema.md` in `feature/akg` with message "Set up MongoDB Atlas with schemas".

- **Initialize AKG (3 hours)**:
  - **Objective**: Build an initial knowledge graph for relational reasoning.
  - **Actions**:
    - Use `networkx` to create a directed graph with nodes (stocks, sectors, macro-indicators) and edges (correlations, influences).
    - Store in `knowledge_graph` collection (`node_id`, `type`, `attributes`, `edges`).
    - Populate with sample data (e.g., `AAPL` → `Tech Sector`, `Tech Sector` → `Interest Rates`).
    - Test graph queries (e.g., shortest path between nodes) in Jupyter.
  - **Outcome**: AKG initialized with sample data for reasoning.
  - **Git**: Commit scripts to `feature/akg` with message "Initialize AKG with networkx".
- **Test MongoDB connection (2 hours)**:
  - **Objective**: Verify connectivity between Jupyter/Colab and MongoDB Atlas.
  - **Actions**:
    - Write a `pymongo` script to connect to the `QuantEdge` cluster using connection string.
    - Insert and query sample AKG nodes (e.g., `AAPL` node).
    - Handle timeouts and errors (e.g., retry logic for network issues).
    - Test in both Jupyter and Colab environments.
  - **Outcome**: Reliable MongoDB connectivity confirmed.
  - **Git**: Commit script to `feature/akg` with message "Test MongoDB connection".
- **Merge branch (1 hour)**:
  - **Objective**: Integrate AKG setup into main repository.
  - **Actions**:
    - Create pull request for `feature/akg` to `main`.
    - Verify CI pipeline passes (tests, linting).
    - Merge after review, close related issues.
  - **Outcome**: AKG setup in `main` branch.
  - **Git**: Push, commit with message "Merge AKG setup to main".

4.3 Week 3: Data Sources and PSS Research

- **Implement yfinance API (3 hours)**:
  - **Objective**: Fetch real-time stock data for AKG and RAG.
  - **Actions**:
    - Write a Python script using `yfinance` to pull 5-minute price data, RSI, and MACD for 20 tickers (e.g., AAPL, MSFT).

- Store data in `stocks` collection (`{ticker, timestamp, price, RSI, MACD}`).
- Update AKG nodes/edges (e.g., `AAPL price` → `Tech Sector influence`).
- Test data retrieval and storage in Jupyter.

- **Outcome**: Stock data integrated into MongoDB and AKG.
- **Git**: Commit script to `feature/data-setup` with message "Implement yfinance API integration".

- **Integrate FRED and X APIs (4 hours)**:
  - **Objective**: Fetch macro-economic and sentiment data for AKG and RAG.
  - **Actions**:
    - Use `fredapi` to fetch interest rates, GDP, and other macro indicators, storing in `macro_data` (`{indicator, value, timestamp}`).
    - Use `tweepy` to fetch X posts for sentiment analysis, storing in `xposts` (`{post_id, text, timestamp}`).
    - Implement caching to manage API limits (FRED: 1000/day, X: 1500 posts/month).
    - Update AKG edges (e.g., `Interest Rates` → `Market Sentiment`).
    - Test data ingestion in Jupyter/Colab.
  - **Outcome**: Macro and sentiment data integrated with caching.
  - **Git**: Commit scripts to `feature/data-setup` with message "Integrate FRED and X APIs".
- **Research PSS and XAI (2 hours)**:
  - **Objective**: Design PSS and XAI features for strategy synthesis and interpretability.
  - **Actions**:
    - Study PSS for dynamic strategy synthesis using RAG, AKG, and user preferences.
    - Research SHAP for XAI to explain LLM outputs.
    - Draft schemas for `strategy_templates` (`{base_strategy, parameters}`) and `interpretations` (`{query, features, weights}`).
    - Document findings in `docs/features.md`.
  - **Outcome**: Feature designs documented for implementation.
  - **Git**: Commit to `docs/features.md` in `feature/data-setup` with message "Document PSS and XAI research".
- **Merge branch (1 hour)**:
  - **Objective**: Integrate data setup into main repository.
  - **Actions**:
    - Create pull request for `feature/data-setup` to `main`.
    - Ensure CI pipeline passes.
    - Merge after review, tag release v0.1.
  - **Outcome**: Data sources and research in `main`.
  - **Git**: Push, commit with message "Merge data setup to main, tag v0.1".

4.4 Week 4: Data Engineering and MLflow

- **Set up MLflow (3 hours)**:
  - **Objective**: Enable experiment tracking for model training and vectorization enhancements.
  - **Actions**:
    - Install `mlflow` in Jupyter/Colab via `pip install mlflow`.
    - Configure a local MLflow server to log parameters, metrics, and artifacts.
    - Test with a dummy model (e.g., logistic regression) to log hyperparameters (e.g., learning rate) and metrics (e.g., accuracy).
    - Document MLflow setup in `docs/mlflow.md`.
  - **Outcome**: MLflow server operational for tracking experiments.
  - **Git**: Commit setup scripts to `feature/mlops` with message "Configure MLflow for experiment tracking".
- **Configure vectorization with enhancements (4 hours)**:
  - **Objective**: Enable efficient vector search for RAG and AKG with optimized embeddings.
  - **Actions**:
    - Use `sentence-transformers/all-mpnet-base-v2` to generate 768-dimensional embeddings for `news`, `xposts`, and `macro_data`.
    - Test lightweight model `all-MiniLM-L6-v2` (384 dimensions) to compare memory usage and retrieval accuracy, logging results in MLflow.
    - Apply dimensionality reduction using `scikit-learn` (PCA) or `umap-learn` (UMAP) to reduce embeddings to 256 dimensions, targeting $<5\%$ loss in retrieval accuracy.
    - Store embeddings in `embeddings` collection with a vector index (e.g., `createIndex({embedding: "2dsphere"})`).
    - Test vector search with sample queries (e.g., "AAPL news sentiment") to verify semantic retrieval.
  - **Outcome**: Vector search enabled with reduced storage footprint and alternative model options.
  - **Git**: Commit scripts and MLflow logs to `feature/vectorization` with message "Configure enhanced vectorization with PCA/UMAP and MiniLM".
- **Set up data pipeline with batch processing (3 hours)**:
  - **Objective**: Automate data ingestion with optimized batch processing for vectorization.
  - **Actions**:
    - Write a Python script using `schedule` to run daily ingestion from yfinance, NewsAPI, FRED, and XAPI.
    - Implement batch processing (e.g., 50 records per batch) to fetch and vectorize data, respecting API limits (NewsAPI: 100/day, X: 1500 posts/month).
    - Store versioning in `data_versions` (`{collection, timestamp, hash}`) to track updates.
    - Test pipeline with a small dataset to ensure data flows to `stocks`, `news`, `xposts`, and

`macro_data`.

- **Outcome**: Efficient data pipeline with batch processing for vectorization.
- **Git**: Commit pipeline scripts to `feature/data-pipeline` with message "Set up data pipeline with batch processing".

- **Merge branch (1 hour)**:
  - **Objective**: Integrate MLflow, vectorization, and pipeline into main repository.
  - **Actions**:
    - Create pull requests for `feature/mlops`, `feature/vectorization`, `feature/data-pipeline` to `main`.
    - Verify CI pipeline passes for all branches.
    - Merge after review, tag release v0.2.
  - **Outcome**: MLflow, vectorization, and pipeline in `main`.
  - **Git**: Push, commit with message "Merge MLflow, vectorization, and pipeline to main, tag v0.2".

5 Phase 2: LLM, RAG, AKG, and PSS Development (Weeks 5–12, July 29–September 23, 2025)

Goal: Fine-tune DistilBERT, develop RAG, AKG, PSS, and model switching for CLE, with enhanced vectorization for efficient data consumption.
Total Hours: 80 hours (10 hours/week).

5.1 Week 5: Data Pipeline Enhancement

- **Enhance ingestion pipeline with incremental updates (4 hours)**:
  - **Objective**: Automate data collection with minimal recomputation for efficiency.
  - **Actions**:
    - Enhance pipeline to fetch, clean, and normalize data from yfinance, NewsAPI, FRED, and XAPI.
    - Check `data_versions` for changes using `timestamp` and `hash` to identify new or updated records.
    - Update `stocks`, `news`, `xposts`, `macro_data`, and AKG nodes/edges only for changed data.
    - Implement retry logic for API failures and log usage in `logs` to track compliance (e.g., XAPI: 1500 posts/month).
    - Test incremental updates with a small dataset (e.g., 10 new X posts).
  - **Outcome**: Robust pipeline with incremental updates for efficiency.
  - **Git**: Commit scripts to `feature/data-pipeline` with message "Enhance pipeline with incremental updates".
- **Vectorize data with enhancements (3 hours)**:
  - **Objective**: Generate optimized embeddings for RAG and AKG.
  - **Actions**:

- Vectorize new/updated text data using `all-mpnet-base-v2` or `all-MiniLM-L6-v2` based on Week 4 results.
- Apply PCA/UMAP to reduce embeddings to 256 dimensions, logging quality metrics (e.g., cosine similarity) in MLflow.
- Batch process data (e.g., 50 records per batch) to optimize compute and API usage.
- Store embeddings in `embeddings` collection and update vector index.
- Test semantic search with sample queries to verify embedding quality.

- **Outcome**: Efficiently stored embeddings with reduced resource usage.
- **Git**: Commit scripts and MLflow logs to `feature/data-pipeline` with message "Vectorize data with PCA/UMAP and batch processing".

- **Test pipeline (2 hours)**:
  - **Objective**: Validate data flow and incremental updates.
  - **Actions**:
    - Run pipeline with a test dataset (e.g., 100 records across collections).
    - Verify updates in `stocks`, `news`, `xposts`, `macro_data`, `embeddings`, and AKG.
    - Check `data_versions` for correct versioning and hash consistency.
    - Test pipeline robustness with simulated API failures.
  - **Outcome**: Validated pipeline with incremental updates and vectorization.
  - **Git**: Commit test scripts to `feature/data-pipeline` with message "Test enhanced pipeline".
- **Merge branch (1 hour)**:
  - **Objective**: Integrate enhanced pipeline into main repository.
  - **Actions**:
    - Create pull request for `feature/data-pipeline` to `main`.
    - Ensure CI pipeline passes (tests, linting).
    - Merge after review, close related issues.
  - **Outcome**: Enhanced pipeline in `main`.
  - **Git**: Push, commit with message "Merge enhanced pipeline to main".

5.2 Week 6: RAG Pipeline Development

- **Implement BM25 (3 hours)**:
  - **Objective**: Enable keyword-based search for RAG pipeline.
  - **Actions**:
    - Use `rank-bm25` to index text fields in `news` and `xposts` collections.
    - Preprocess text (e.g., tokenization, stop-word removal) to optimize BM25 scoring.
    - Test retrieval accuracy with sample queries (e.g., "AAPL earnings"), targeting >85% recall.
    - Log metrics (recall, latency) in MLflow for comparison with vector search.

- **Outcome**: Functional BM25 search for keyword queries.
- **Git**: Commit scripts to `feature/rag` with message "Implement BM25 for keyword search".

- **Implement FAISS with hybrid tuning (3 hours)**:
  - **Objective**: Enable optimized vector search with tuned hybrid retrieval.
  - **Actions**:
    - Use `faiss-cpu` to index `embeddings` collection for similarity search.
    - Test search with sample queries, targeting >90% precision.
    - Fine-tune BM25/FAISS balance (e.g., 30% BM25, 70% FAISS for semantic queries, 70% BM25 for keyword-heavy queries).
    - Log retrieval metrics (precision, recall, latency) in MLflow to determine optimal weighting.
    - Store optimal parameters in `rag` configuration file.
  - **Outcome**: FAISS vector search with tuned hybrid retrieval.
  - **Git**: Commit scripts and MLflow logs to `feature/rag` with message "Implement FAISS with hybrid BM25 tuning".
- **Integrate RAG with AKG (3 hours)**:
  - **Objective**: Combine keyword/vector search with graph reasoning for RAG.
  - **Actions**:
    - Develop a hybrid RAG pipeline to retrieve text via BM25/FAISS and augment with AKG paths (e.g., `AAPL` → `Tech Sector` → `Interest Rates`).
    - Implement query processing to combine retrieved documents with AKG-derived context.
    - Test RAG output with sample queries (e.g., "Should I buy AAPL?") to ensure coherent augmentation.
    - Log RAG performance (e.g., relevance, latency) in MLflow.
  - **Outcome**: Operational RAG pipeline with AKG integration.
  - **Git**: Commit scripts to `feature/rag` with message "Integrate RAG with AKG".
- **Merge branch (1 hour)**:
  - **Objective**: Integrate RAG pipeline into main repository.
  - **Actions**:
    - Create pull request for `feature/rag` to `main`.
    - Verify CI pipeline passes.
    - Merge after review, tag release v0.3.
  - **Outcome**: RAG pipeline in `main`.
  - **Git**: Push, commit with message "Merge RAG pipeline to main, tag v0.3".

5.3 Weeks 7–8: DistilBERT Fine-Tuning

- **Create fine-tuning dataset (8 hours)**:
  - **Objective**: Prepare a high-quality dataset for DistilBERT fine-tuning.

- **Actions**:
  - Curate Alpaca-style JSON dataset from `chat_history`, `user_preferences`, `stocks`, `news`, `xposts`, `macro_data`, and `feedback`.
  - Structure dataset as `{instruction, input, output}` (e.g., `instruction`: "Recommend a stock", `input`: "User prefers low risk", `output`: "Buy JNJ").
  - Validate dataset for completeness and format, ensuring 500+ samples.
  - Store in `datasets` collection.
- **Outcome**: Fine-tuning dataset ready for DistilBERT.
- **Git**: Commit dataset to `feature/llm-finetune` with message "Create fine-tuning dataset".

- **Fine-tune DistilBERT (8 hours)**:
  - **Objective**: Train DistilBERT for financial queries.
  - **Actions**:
    - Use `transformers.peft` with QLoRA (4-bit quantization) on Colab T4 GPU.
    - Fine-tune DistilBERT on curated dataset, optimizing for financial query responses.
    - Log hyperparameters (e.g., learning rate, epochs) and metrics (F1-score >80%) in MLflow.
    - Save model checkpoints to `model_config`.
  - **Outcome**: Fine-tuned DistilBERT model for QuantEdge.
  - **Git**: Commit model config to `feature/llm-finetune` with message "Fine-tune DistilBERT with QLoRA".
- **Test LLM (4 hours)**:
  - **Objective**: Validate DistilBERT performance with RAG/AKG context.
  - **Actions**:
    - Test model on sample queries (e.g., "Analyze AAPL trend") using RAG/AKG context.
    - Measure F1-score (>80%) and latency (<1s) for 50+ queries.
    - Log results in MLflow and identify any failure cases.
    - Adjust prompts if needed to improve response quality.
  - **Outcome**: Validated LLM performance.
  - **Git**: Commit test scripts to `feature/llm-finetune` with message "Test fine-tuned DistilBERT".

5.4 Weeks 9–10: PSS Implementation

- **Develop PSS (8 hours)**:
  - **Objective**: Enable dynamic synthesis of custom trading strategies.
  - **Actions**:
    - Define `strategy_templates` (`{base_strategy, parameters}`) for strategies like Momentum, Mean Reversion.
    - Develop LLM logic to blend strategies using RAG/AKG outputs and `user_preferences`.
    - Test synthesis with sample inputs (e.g., combine Momentum and Mean Reversion for a

user preferring high returns).
    - Store synthesized strategies in `strategies` collection.
  - **Outcome**: Functional PSS for custom strategies.
  - **Git**: Commit scripts to `feature/pss` with message "Develop PSS for strategy synthesis".
- **Test PSS (6 hours)**:
  - **Objective**: Validate PSS-generated strategies.
  - **Actions**:
    - Backtest strategies using `backtrader` on historical data (e.g., 6 months of AAPL prices).
    - Compare Sharpe ratio (target 1–1.5) and other metrics (e.g., drawdown).
    - Store results in `strategies` collection.
    - Log performance in MLflow for analysis.
  - **Outcome**: Validated PSS with robust strategies.
  - **Git**: Commit test scripts to `feature/pss` with message "Test PSS strategies with backtrader".
- **Merge branch (2 hours)**:
  - **Objective**: Integrate PSS into main repository.
  - **Actions**:
    - Create pull request for `feature/pss` to `main`.
    - Verify CI pipeline passes.
    - Merge after review, close issues.
  - **Outcome**: PSS in `main`.
  - **Git**: Push, commit with message "Merge PSS to main".

5.5 Weeks 11–12: Model Switching and CLE

- **Implement model switching (8 hours)**:
  - **Objective**: Enable dynamic loading of LLMs for flexibility.
  - **Actions**:
    - Develop a modular interface using `transformers.AutoModel` and `AutoTokenizer`.
    - Store configurations in `model_config` (`{model_name, path, tokenizer}`).
    - Test switching between DistilBERT and placeholders for Llama-3/Mixtral.
    - Verify model loading and inference in Colab with T4 GPU.
  - **Outcome**: Modular LLM interface for dynamic switching.
  - **Git**: Commit scripts to `feature/model-switching` with message "Implement LLM model switching".
- **Set up CLE (8 hours)**:
  - **Objective**: Automate continuous model improvement.
  - **Actions**:

- Automate biweekly fine-tuning with QLoRA/DPO using `datasets` and `feedback`.
- Schedule AKG updates to reflect new data relationships.
- Log model performance (e.g., F1-score, latency) in MLflow.
- Update `model_config` if a better model is found.

- **Outcome**: Functional CLE pipeline for ongoing improvement.
- **Git**: Commit scripts to `feature/cle` with message "Set up CLE for continuous learning".

- **Merge and release (4 hours)**:
  - **Objective**: Finalize Phase 2 with integrated components.
  - **Actions**:
    - Create pull requests for `feature/llm-finetune`, `feature/model-switching`, `feature/cle` to `main`.
    - Verify CI pipeline passes.
    - Merge after review, tag release v0.4.
  - **Outcome**: LLM pipeline in `main`.
  - **Git**: Push, commit with message "Merge LLM, model switching, and CLE to main, tag v0.4".

6 Phase 3: Strategy, Portfolio, and XAI Development (Weeks 13–18, September 24–November 4, 2025)

Goal: Implement strategies, simulate portfolios, and integrate XAI with multi-modal outputs.
Total Hours: 60 hours (10 hours/week).

6.1 Weeks 13–15: Strategy and Portfolio Simulation

- **Enhance strategies with PSS (10 hours)**:
  - **Objective**: Refine PSS for robust strategy synthesis.
  - **Actions**:
    - Use `backtrader` to backtest PSS-generated strategies (Momentum, Mean Reversion, Statistical Arbitrage).
    - Update `strategies` collection with results (`{strategy_id, parameters, performance}`).
    - Enhance AKG edges with strategy correlations (e.g., `Momentum` → `High Volatility Stocks`).
    - Test strategies with diverse user preferences (e.g., risk-averse vs. aggressive).
  - **Outcome**: Enhanced PSS strategies with AKG integration.
  - **Git**: Commit scripts to `feature/strategies` with message "Enhance PSS strategies with backtrader".
- **Simulate portfolios (8 hours)**:
  - **Objective**: Create simulated portfolios for user interaction.
  - **Actions**:
    - Simulate trades using `backtrader` with PSS strategies.

- Store results in `portfolios` (`{user_id, assets, weights, trades}`).
- Generate Plotly charts (e.g., portfolio allocation pie chart) and store HTML in `chat_history`.
- Test portfolio simulations with sample user inputs.

- **Outcome**: Functional simulated portfolios with visualizations.
- **Git**: Commit scripts to `feature/portfolio` with message "Implement portfolio simulation".

- **Integrate with LLM (6 hours)**:
  - **Objective**: Enable LLM-driven portfolio queries.
  - **Actions**:
    - Develop LLM queries for portfolio review/update (e.g., "Adjust for low risk") using RAG/AKG context.
    - Store updates in `portfolios` collection.
    - Test query responses for accuracy and coherence.
    - Log performance in MLflow.
  - **Outcome**: LLM-integrated portfolio functionality.
  - **Git**: Commit scripts to `feature/portfolio` with message "Integrate LLM with portfolio queries".

6.2 Weeks 16–18: XAI Implementation

- **Implement SHAP (8 hours)**:
  - **Objective**: Ensure interpretable LLM outputs.
  - **Actions**:
    - Apply `shap` to DistilBERT outputs to compute feature importance for queries.
    - Store results in `interpretations` (`{query, features, weights}`).
    - Test SHAP explanations with sample queries (e.g., "Why buy AAPL?").
    - Log explanation quality in MLflow.
  - **Outcome**: Interpretable LLM outputs with SHAP.
  - **Git**: Commit scripts to `feature/xai` with message "Implement SHAP for XAI".
- **Generate AKG explanations (8 hours)**:
  - **Objective**: Provide natural language explanations using AKG.
  - **Actions**:
    - Use AKG paths to generate explanations (e.g., "AAPL: buy due to positive X sentiment and low interest rates").
    - Integrate explanations with LLM responses for user queries.
    - Test explanations for clarity and relevance with 50+ queries.
    - Store in `interpretations` collection.
  - **Outcome**: Functional AKG-based explanations.

- **Git**: Commit scripts to `feature/xai` with message "Generate AKG-based explanations".

- **Merge and release (4 hours)**:
  - **Objective**: Finalize Phase 3 with integrated components.
  - **Actions**:
    - Create pull requests for `feature/strategies`, `feature/portfolio`, `feature/xai` to `main`.
    - Verify CI pipeline passes.
    - Merge after review, tag release v0.5.
  - **Outcome**: Strategies, portfolios, and XAI in `main`.
  - **Git**: Push, commit with message "Merge strategies, portfolio, and XAI to main, tag v0.5".

7 Phase 4: Multi-Modal Dashboard Development (Weeks 19–22, November 5–December 2, 2025)

Goal: Build a Jupyter-based dashboard with multi-modal interface and export functionality.
Total Hours: 40 hours (10 hours/week).

- **Develop dashboard (12 hours)**:
  - **Objective**: Create an interactive user interface for strategy and portfolio management.
  - **Actions**:
    - Use `ipywidgets` to create forms for strategy input, portfolio review, and LLM queries.
    - Generate `plotly` charts (e.g., stock trends, portfolio allocation).
    - Implement PDF/CSV export using `pdfkit` and `pandas`.
    - Integrate multi-modal LLM responses (text, charts) in the dashboard.
  - **Outcome**: Functional dashboard with multi-modal outputs.
  - **Git**: Commit scripts to `feature/dashboard` with message "Develop multi-modal dashboard".

- **Test dashboard (8 hours)**:
  - **Objective**: Validate dashboard usability and performance.
  - **Actions**:
    - Test forms, charts, and exports in Jupyter/Colab.
    - Collect user `feedback` on usability.
    - Measure response time (<2s) for 50+ interactions.
    - Log results in MLflow.
  - **Outcome**: Usable and responsive dashboard.
  - **Git**: Commit test scripts to `feature/dashboard` with message "Test dashboard functionality".

- **Merge and release (4 hours)**:
  - **Objective**: Finalize dashboard integration.

- **Actions**:
  - Create pull request for `feature/dashboard` to `main`.
  - Verify CI pipeline passes.
  - Merge after review, tag release v0.6.
- **Outcome**: Dashboard in `main`.
- **Git**: Push, commit with message "Merge dashboard to main, tag v0.6".

8 Phase 5: Testing, Optimization, and Compliance (Weeks 23–27, December 3, 2025–January 13, 2026)

Goal: Test system robustness, optimize MongoDB, and ensure ECF compliance.
Total Hours: 50 hours (10 hours/week).

- **Unit and integration tests (12 hours)**:
  - **Objective**: Ensure system functionality across components.
  - **Actions**:
    - Write `pytest` tests for pipeline, RAG, AKG, PSS, LLM, dashboard, and model switching.
    - Test MongoDB queries (e.g., vector search, AKG traversal).
    - Target 90% test coverage using `pytest-cov`.
    - Log test results in MLflow.
  - **Outcome**: Comprehensive test suite with high coverage.
  - **Git**: Commit tests to `feature/testing` with message "Implement unit and integration tests".
- **Robustness tests (8 hours)**:
  - **Objective**: Validate system under adverse conditions.
  - **Actions**:
    - Test LLM/PSS with incomplete data (e.g., partial X posts), targeting F1-score >80%.
    - Test AKG with missing edges to ensure graceful degradation.
    - Simulate high query loads to test MongoDB performance.
    - Log robustness metrics in MLflow.
  - **Outcome**: Robust system under noise and stress.
  - **Git**: Commit tests to `feature/testing` with message "Conduct robustness tests".
- **Optimize MongoDB (8 hours)**:
  - **Objective**: Improve database performance within M0 limits.
  - **Actions**:
    - Add compound indexes to `embeddings`, `stocks`, and `knowledge_graph`.
    - Prune data older than 30 days, archiving to JSON in GitHub.
    - Optimize AKG queries to <100ms using index strategies.
    - Test query performance with sample workloads.

- **Outcome**: Optimized MongoDB performance.
- **Git**: Commit scripts to `feature/optimization` with message "Optimize MongoDB with indexes and pruning".

- **Implement ECF (8 hours)**:
  - **Objective**: Ensure compliance with API and ethical standards.
  - **Actions**:
    - Monitor API usage in `logs` to stay within limits (e.g., NewsAPI: 100/day).
    - Anonymize data in `user_preferences` and `chat_history`.
    - Flag non-compliant queries in `compliance_flags`.
    - Document compliance measures in `docs/compliance.md`.
  - **Outcome**: ECF-compliant system.
  - **Git**: Commit scripts to `feature/compliance` with message "Implement ECF for compliance".
- **Merge and release (4 hours)**:
  - **Objective**: Finalize Phase 5 with integrated components.
  - **Actions**:
    - Create pull requests for `feature/testing`, `feature/optimization`, `feature/compliance` to `main`.
    - Verify CI pipeline passes.
    - Merge after review, tag release v0.7.
  - **Outcome**: Optimized, compliant system in `main`.
  - **Git**: Push, commit with message "Merge testing, optimization, and compliance to main, tag v0.7".

9 Phase 6: Demo, Documentation, and Finalization (Weeks 28–30, January 14–February 3, 2026)

Goal: Create a demo notebook, finalize documentation, and submit FYP.
Total Hours: 30 hours (10 hours/week).

- **Create demo notebook (10 hours)**:
  - **Objective**: Showcase QuantEdge system functionality.
  - **Actions**:
    - Develop a Jupyter/Colab notebook demonstrating setup, pipeline, RAG, AKG, PSS, XAI, dashboard, and model switching.
    - Include multi-modal outputs (text, Plotly charts, PDF exports).
    - Test demo in both Jupyter and Colab environments.
    - Ensure reproducibility with `environment.yml`.
  - **Outcome**: Comprehensive demo notebook.
  - **Git**: Commit notebook to `feature/demo` with message "Create demo notebook".

- **Finalize documentation (12 hours)**:
  - **Objective**: Document project for users and evaluators.
  - **Actions**:
    - Write `README.md` with project overview, setup, and usage.
    - Create user guide, API usage, and details on AKG, PSS, XAI, CLE, ECF, and vectorization enhancements in `docs/`.
    - Include setup instructions for Jupyter/Colab and MongoDB.
    - Document compliance measures and vectorization optimizations.
  - **Outcome**: Comprehensive documentation.
  - **Git**: Commit to `docs/` with message "Finalize project documentation".
- **Final release (8 hours)**:
  - **Objective**: Release and submit QuantEdge project.
  - **Actions**:
    - Create pull request for `feature/demo` to `main`.
    - Verify CI pipeline passes.
    - Merge after review, tag release v1.0.
    - Prepare FYP submission with demo notebook and documentation.
  - **Outcome**: Project complete and submitted.
  - **Git**: Push, commit with message "Merge demo to main, tag v1.0".

10 Key Milestones

- **July 28, 2025**: Jupyter/Colab, MongoDB with AKG, GitHub, and data ingestion with enhanced vectorization (PCA/UMAP, MiniLM, batch processing) set up. Tag v0.2.
- **September 23, 2025**: DistilBERT fine-tuned, RAG with hybrid BM25/FAISS tuning, AKG, PSS, model switching, and CLE implemented. Tag v0.4.
- **November 4, 2025**: Strategies, portfolios, and XAI with multi-modal outputs complete. Tag v0.5.
- **December 2, 2025**: Multi-modal Jupyter dashboard with export functionality ready. Tag v0.6.
- **January 13, 2026**: System tested, optimized, and ECF-compliant. Tag v0.7.
- **February 3, 2026**: Final demo, documentation, and FYP submitted. Tag v1.0.

11 Git/GitHub Workflow

- **Branching**: Use `main` for production-ready code, `feature/*` for tasks (e.g., `feature/akg`, `feature/rag`).
- **Commits**: Use clear messages (e.g., "Add AKG node creation script"). Use `nbdime` for notebook diffs.
- **Pull Requests**: Create PRs for `feature/*` branches, merge to `main` after CI passes and review.

- **CI Pipeline**: GitHub Actions runs `pytest`, `flake8`, and notebook execution on push.
- **Releases**: Tag milestones (e.g., v0.1, v1.0) in GitHub Releases.
- **Issue Tracking**: Create issues for tasks, link to PRs, close upon merge.

12 Notes

- **Jupyter/Colab**: Jupyter for persistent development, Colab for GPU tasks (T4 GPU, 16GB VRAM). Sync `.ipynb` with `nbdime`.
- **MongoDB M0**: 512MB limit requires compound indexes, pruning (>30 days), and JSON archiving. Use static IP for Jupyter, `0.0.0.0/0` for Colab.
- **API Limits**: yfinance (unlimited), NewsAPI (100/day), FRED (1000/day), XAPI (1500 posts/month). Cache in MongoDB, log usage in `logs`.
- **LLM**: DistilBERT with QLoRA/DPO, switchable to Llama-3/Mixtral via `model_config`. Use SHAP for XAI, dynamic prompts for responses.
- **AKG**: Graph-based reasoning in `knowledge_graph`, updated daily for scalability.
- **PSS**: Custom strategies in `strategy_templates`, synthesized via LLM and AKG.
- **CLE**: Biweekly fine-tuning with DPO, automated ingestion via `schedule`.
- **ECF**: Compliance via `logs`, `compliance_flags`, anonymized data in `user_preferences`, `chat_history`.
- **Security**: Store API keys in environment variables or Colab Secrets.
- **Vectorization Enhancements**:
  - **Dimensionality Reduction**: PCA/UMAP to reduce embeddings to 256 dimensions, minimizing storage needs.
  - **Incremental Updates**: Vectorize only new/updated data to reduce compute and API usage.
  - **Lightweight Models**: Test `all-MiniLM-L6-v2` for lower resource consumption.
  - **Batch Processing**: Process data in batches (e.g., 50 records) for efficient vectorization.
  - **Hybrid Search Tuning**: Optimize BM25/FAISS balance for diverse query types, logged in MLflow.