Project Architecture 1.0

ERAU Graph-Theory Project

Phase 1: Database and Training Data Creation

Tools and Steps

1. **Database Creation:**
    - **Tool:** Neo4j (Graph Database)
        - Why: Optimized for graph-theoretic data storage, easy to query with Cypher for graph relationships.
    - **Alternative:** ArangoDB (Multi-Model DB) if hybrid storage (key-value + graph) is required.
    - **Task:** Build a schema with nodes (atoms) and edges (bonds) that represent molecular structures. Include fields for material properties extracted from **Ansys Granta** and **external molecular datasets (e.g., PubChem, Materials Project).**
2. **Data Preprocessing:**
    - **Tool:** Python with RDKit and Pandas
        - Why: RDKit converts SMILES/InChI into graph representations and calculates descriptors, while Pandas efficiently processes tabular data.
    - **Task:** Standardize data into molecular graphs, clean datasets, and compute node/edge feature vectors.
3. **Data Storage and Pipelines:**
    - **Tool:** Apache Airflow or Prefect
        - Why: Automates ETL (Extract, Transform, Load) processes and ensures smooth data ingestion into the database.
    - **Task:** Set up workflows for periodic data updates from Ansys Granta and external sources.

Phase 2: Model Development

Tools and Steps

1. **Model Framework:**
    - **Tool:** PyTorch Geometric or TensorFlow Graph Neural Networks
        - Why: Optimized for working with graph-structured data.
    - **Task:** Develop base models such as Graph Convolutional Networks (GCNs) for predicting molecular properties.
2. **Training Pipeline:**
    - **Tool:** MLflow
        - Why: Manages the lifecycle of experiments, including versioning and hyperparameter tuning.
    - **Task:** Train baseline models using the molecular graph database, with features engineered via RDKit or cheminformatics libraries.
3. **Evaluation Metrics:**
    - **Tool:** Scikit-learn or custom metrics
        - Why: Provides standard regression and classification evaluation metrics.
    - **Task:** Evaluate model performance using metrics like RMSE, MAE, or R^2.

Phase 2A: Evolution of Materials Model

Tools and Steps

1. **Material Evolution Training:**
    - **Tool:** Sequence-to-Sequence Models (e.g., Transformer or LSTM with PyTorch)
        - Why: Captures temporal or sequential relationships in material evolution.
    - **Task:** Train a model to map precursor materials (e.g., graphite) to evolved materials (e.g., graphene) based on historical data.
2. **Data Augmentation:**
    - **Tool:** Python with Material-Specific Libraries (e.g., Matminer)

        - Why: Augments training data with additional insights into properties of materials at various evolutionary stages.
    - **Task:** Include phase transitions, processing techniques, and historical advancements as features.
3. **Complex Materials Evolution:**
    - **Tool:** Graph Neural Networks with Temporal Graph Layers
        - Why: Allows modeling of time-evolving graphs, such as the progression from graphite composites to carbon fiber-reinforced polymers.
    - **Task:** Extend models to predict evolutionary steps for complex materials.

Phase 2B: Hidden Connections Materials Model

Tools and Steps

1. **Correlation Analysis:**
    - **Tool:** PCA/T-SNE + Cluster Analysis (Scikit-learn)
        - Why: Identifies latent patterns in high-dimensional data.
    - **Task:** Analyze material features to discover unusual correlations (e.g., metals behaving like rubber).
2. **Deep Learning for Hidden Patterns:**
    - **Tool:** Autoencoders or Contrastive Learning (PyTorch/TensorFlow)
        - Why: Encodes material features into latent spaces where hidden relationships are easier to detect.
    - **Task:** Train models to uncover unexpected relationships between seemingly unrelated materials.
3. **Graph-Level Pattern Recognition:**
    - **Tool:** Graph Attention Networks (GATs)
        - Why: Focuses on important parts of the graph, identifying key substructures and their roles in material behavior.

- **Task:** Highlight graph regions associated with anomalous or unexplained properties.

Phase 3: Database Query Interface/AI Model

Tools and Steps

1. **Natural Language Interface:**
    - **Tool:** Llama API (Llama-3 or similar models fine-tuned for material science queries)
        - Why: Enables users to ask natural language questions about the database.
        - Why: Possible offline capabilities for desktop app integration
    - **Task:** Fine-tune a language model using FAQs, query patterns, and expert annotations in material science.
2. **Graph Query Integration:**
    - **Tool:** Neo4j Integration with GPT via Cypher Query API
        - Why: Combines natural language processing with graph-based query responses.
    - **Task:** Map user queries to Cypher statements that fetch graph data and provide structured responses.
3. **Recommendation System:**

    - **Tool:** Collaborative Filtering or Knowledge Graph-Based Recommendations
        - Why: Provides high-quality material suggestions based on user requirements.
    - **Task:** Implement a recommendation system trained on user-query interaction data to suggest optimal materials for specific applications.

Simplified Tool Selection Summary:

Phase	Key Tool(s)	Rationale
Phase 1	Neo4j, RDKit, Apache Airflow	Efficient graph storage, molecular data processing, and automated workflows.
Phase 2	PyTorch Geometric, MLflow	Scalable, robust framework for GNN-based model development and tracking.
Phase 2A	Transformer Models, Matminer	Captures material evolution and complex transitions effectively.
Phase 2B	GATs, Autoencoders, Contrastive Learning	Identifies hidden correlations and patterns in materials science.
Phase 3	Lamma-3 API, Neo4j, Graph-Based Recommendations	Enables user-friendly queries and high-quality material recommendations.