1.8 Methodology

The proposed project, **CodeCodez**, adopts a structured, multi-phase methodology that blends hierarchical task decomposition, graph-based orchestration, and large language model (LLM) specialization to generate fully functional software projects from natural language input. This methodology ensures that the generated systems are not only syntactically accurate but also structurally sound, maintainable, and production-ready.

1. Task Decomposition using Tree-Based Planning

The first step in the methodology involves interpreting the user's natural language requirement and decomposing it into logically distinct subtasks. This decomposition is performed using a hierarchical **tree-based planning approach**, where the main project goal is treated as the root, and various modules such as frontend, backend, database, authentication, testing, and documentation are treated as nodes in the tree.

Each node can further branch into subcomponents—e.g., the backend can split into routing, controller logic, and data access layers—ensuring the breakdown is as granular as required. This strategy mirrors how human developers conceptualize complex systems and allows for efficient parallel processing in later stages.

2. Directed Acyclic Graph (DAG) Conversion for Dependency Mapping

Once the tree is constructed, it is converted into a **Directed Acyclic Graph (DAG)** to manage the order of task execution and inter-component dependencies. This step ensures that modules dependent on others (e.g., UI relying on backend APIs, or database setup preceding ORM integration) are generated in the correct sequence.

This graph-based representation enables safe, parallel generation where feasible, reducing computation time while preserving logical consistency across modules.

3. Multi-Agent LLM Allocation

Each subtask from the DAG is routed to a **specialized LLM agent**, selected based on its domain expertise. For example:

- UI-related tasks are sent to LLMs fine-tuned for React or HTML/CSS.