METHODOLOGY ADOPTED

3.1 Investigative Techniques

S.No.	Investigative Projects Techniques	Investigative Techniques Description	Investigative Projects Examples
1	Literature-Driven Investigation	A thorough survey of recent academic literature was carried out to study the latest developments in large language model orchestration, hierarchical task decomposition, and collaborative reasoning mechanisms. The investigation highlighted how multiple specialized LLMs can be combined to handle different subtasks such as frontend, backend, and testing. This method ensured that CodeCodez builds upon validated frameworks while addressing gaps in scalability, interpretability, and practical deployment of multi-agent LLMs.	Chain-of-Agents [1], Corex [2], HALO [3], Mixture-of-Agents [4], Survey on Multi-Agent Systems [10].
2	Requirement-Oriented Investigation	Requirements were gathered through surveys, developer interviews, and exploratory workshops. This technique enabled identifying pain points such as repetitive scaffolding, manual configuration, and fragmented documentation practices. By grounding the system in real-world challenges faced by developers, the investigative method ensured CodeCodez is practically useful and compatible with standard workflows like Git versioning, CI/CD pipelines, and Docker-based deployment.	Developer workflow studies; ReConcile [6], Multi-Agent Debate [7].
3	Experimental Prototyping	Rapid prototyping was employed to validate feasibility of task decomposition and specialized LLM roles. Hierarchical task trees and Directed Acyclic Graphs (DAGs) were used to break down software requirements. Multiple iterations tested prompt engineering, error correction loops, and role-based division of labor. This technique helped identify bottlenecks such as hallucination in generated code and inefficiencies in agent collaboration, which were mitigated by adopting strategies like pruning redundant agents and using lightweight execution with Ollama.	Tree-of-Code [13], MAEL [9], Context Engineering [17], AdaCoder [16].
4	Comparative Tool Analysis	A comparative study of different tools and frameworks was conducted to select the most adaptable architecture. LangChain was chosen for orchestration due to its modularity, while embedding models and vector databases were evaluated for retrieval tasks. Investigating across multiple frameworks also highlighted methods like agent pruning, reasoning aggregation, and adaptive orchestration, which ensured CodeCodez remains scalable and future-ready.	LangChain, AgentDropout [8], Mixture-of-Agents [4], Survey on Multi-Agent Systems [10].
5	Scenario-Based Validation	Scenario-based investigation tested CodeCodez against diverse project categories such as web applications, machine learning pipelines, and analytics dashboards. This approach ensured robustness and adaptability across domains. The results revealed that while frontend scaffolding was efficiently automated, ML pipelines required additional support for preprocessing and dependency management. Insights from these scenarios	DART-LLM [18], Survey on Code Generation [19].