		refined agent specialization and validated the multi-agent design of CodeCodez.	
6	Ethical and Practical Considerations	An investigative focus on ethics was employed to address challenges such as bias propagation, lack of explainability, and potential misuse of generated code. Round-table consensus mechanisms and debate-style reasoning were explored to ensure factuality and fairness in outputs. Embedding responsible AI practices within the architecture helped build a system that is not only technically robust but also socially responsible.	ReConcile [6], Multi-Agent Debate [7], Belief-driven Reasoning [5].

TABLE 7: Investigative Techniques

3.2 Proposed Solution

The proposed solution, CodeCodez, is a multi-agent large language model (LLM) framework designed to automate the generation of production-ready software projects from natural language specifications. The system builds on recent advances in multi-agent orchestration, reasoning, and code generation, while addressing gaps in scalability, interpretability, and developer integration identified in the literature.

The solution is structured around four pillars: hierarchical task decomposition, role-specialized agents, collaborative reasoning mechanisms, and iterative validation through testing and self-reflection. Together, these elements enable CodeCodez to generate modular, scalable, and well-documented software systems that align with real-world engineering practices.

Hierarchical Task Decomposition

Central to the design of CodeCodez is hierarchical task decomposition. Inspired by *HALO* [3], *TDAG* [12], and *DART-LLM* [18], tasks are first broken down into high-level components—such as system architecture, frontend development, backend development, testing, and documentation—before being further subdivided into granular subtasks.

This structured decomposition mitigates the limitations of linear pipelines (e.g., standard Chains of Thought [11]) by ensuring that tasks are processed in parallel, with explicit dependencies modeled as directed graphs. For instance, database schema design must precede API implementation, while unit test creation can proceed in parallel with module coding.

Unlike flat single-agent systems, the hierarchical design provides:

- Scalability — enabling the system to manage projects of varying complexity.