CodeCodez has been conceived as a holistic solution through which the complete software development process—from detailed project specifications to a fully functional, maintainable codebase—is automated. Comprehensive project descriptions are transformed by CodeCodez into structured outputs that include well-organized directories, configuration files, and modular code components. This integrated approach ensures that the generated code not only meets immediate functional requirements but is also designed with scalability and ease of maintenance in mind. In addition, the risk of bugs and inconsistencies is significantly reduced, enhancing the overall reliability of the software.

By bridging the gap between rapid code generation and the creation of high-quality, maintainable software, production costs are reduced, and overall software quality is improved. A new standard is set by CodeCodez, enabling organizations to meet the demands of modern software development more efficiently and effectively.

1.3 Research Gaps

Despite the rapid advancements in large language models (LLMs) and their collaborative frameworks, several gaps persist in their application to fully autonomous and structurally reliable software generation. Through a review of state-of-the-art systems and approaches, the following key research gaps have been identified:

- **Lack of End-to-End Autonomous Project Generation**

While many systems excel in generating modular code snippets, they do not address the full software development lifecycle—from directory scaffolding to test cases and deployment files. For instance, solutions like AgentCoder [15] and AdaCoder [16] introduce multi-agent setups for iterative code generation, but still rely heavily on human prompts for error correction, environment configuration, and final integration. This gap highlights the need for platforms that take structured input and autonomously generate complete, production-ready codebases.

- **Insufficient Integration of Task Decomposition with LLM Workflows**

Although task decomposition has been explored using graph-based or tree-based methods (e.g., [12], [13]), many approaches fail to fully integrate these structures into the LLM inference process. The planning mechanisms often remain external, manually specified, or static. There is a pressing need for systems where **task decomposition dynamically informs prompt design, execution**