tracking of generation pipelines—if one node fails or produces inconsistent results, only that portion of the pipeline is revisited and regenerated.

Specialization Through Multiple LLM Agents

Unlike traditional code assistants that rely on a single generic LLM, CodeCodez deploys **multiple LLMs**, each specialized for a domain. For example, one agent focuses on Python scripting, another on React-based UI generation, a third on test suite design, and yet another on documentation formatting. This multi-agent architecture reflects how software teams function in real scenarios, where different experts handle different roles.

By assigning tasks to specialized agents, the platform reduces logical inconsistencies, improves semantic coherence, and enables the generation of components that are structurally compatible and stylistically consistent. The **orchestration layer** supervises this collaboration, manages communication between models, and ensures smooth fallback and re-prompting in case of errors or inconsistencies.

Built-in Testing and Error Handling

One of the most innovative aspects of CodeCodez is its built-in **testing and correction loop**. After generating code for each subtask, the system simultaneously generates and executes test cases using appropriate frameworks like Pytest for Python, Jest for JavaScript, or Postman for APIs. If a test fails, the error logs are parsed and fed back into the system, prompting the respective LLM agent to revise and regenerate the faulty code section.

This iterative loop of **test → fail → correct → verify** allows the system to move closer to production-level reliability. By catching bugs at generation time rather than post-deployment, it dramatically reduces debugging overhead and aligns generated code with expected behavior and performance.

Context-Aware Documentation Generation

Readability and maintainability are essential in any software project, especially when handed off to a new team. CodeCodez ensures that all generated modules are accompanied by **auto-generated documentation**. This includes markdown files with explanations of each module's purpose, input-output signatures, internal dependencies, and setup instructions.