# Technical Architecture Summary

## Framework

- \*\*LangGraph's GraphChain\*\*:  
 - Framework for building workflows with modular agents as nodes.  
 - Nodes execute in an order defined by a dependency graph.  
 - Enables flexibility, scalability, and clear task separation.

## Techniques and Methodologies

1. \*\*Agent-Oriented Design\*\*:  
 - Business logic is encapsulated within \*\*GraphChain Nodes\*\*, where each agent is responsible for a single task.  
 - Agents are modular and can be individually tested or reused.  
  
2. \*\*Dependency Management\*\*:  
 - Task flow is explicitly defined using the `add\_parent()` method.  
 - For example:  
 - The \*\*Decision Agent\*\* depends on the outputs of the \*\*Eligibility Agents\*\*.  
 - The \*\*Adjustment Agent\*\* depends on the Decision Agent.  
  
3. \*\*Data Preprocessing for Accurate Comparisons\*\*:  
 - All input files are preprocessed to ensure consistency:  
 - \*\*Date Normalization\*\*: Dates converted to `%Y-%m-%d` format.  
 - \*\*Text Standardization\*\*: All text columns (e.g., `member\_name`, `provider\_name`) are trimmed and converted to lowercase.  
 - \*\*Numeric Precision\*\*: All numeric columns (e.g., `Allowed Amount`, `Total Charged`) are converted to floats for precise comparisons.  
  
4. \*\*Asynchronous Execution\*\*:  
 - Achieved using Python’s `asyncio`:  
 - Multiple agents execute concurrently, improving efficiency.  
 - For instance:  
 - \*\*MemberEligibilityNode\*\*, \*\*ProviderEligibilityNode\*\*, and \*\*ProviderNetworkNode\*\* are executed in parallel using `asyncio.gather()`.

## Agents and Their Roles

#### \*\*1. Eligibility Agents\*\*  
- \*\*MemberEligibilityNode\*\*:  
 - Reads and validates `member\_name` and `service\_date` against the Membership File.  
 - Ensures the `service\_date` falls within `effective\_date` and `termination\_date`.  
  
- \*\*ProviderEligibilityNode\*\*:  
 - Checks the `provider\_name` and contract dates (`Contract\_from`, `Contract\_to`) in the Provider File.  
 - Ensures the provider is active during the service period.  
  
- \*\*ProviderNetworkNode\*\*:  
 - Verifies that the `provider\_name` has an "active" status in the Provider Network File.  
  
#### \*\*2. Decision Agent\*\*  
- \*\*DecisionNode\*\*:  
 - Compares claim details from the summarized input file with records in the Claims File.  
 - Key validation criteria:  
 - `Total Charged`, `Allowed Amount`, and `Patient Responsibility` must match.  
 - Checks if `Elevance Paid` is less than `Elevance Responsibility`.  
 - Outputs:  
 - \*\*"Adjust"\*\*: Indicates an adjustment is required.  
 - \*\*"No Adjust"\*\*: Indicates no adjustment is needed.  
  
#### \*\*3. Adjustment Agent\*\*  
- \*\*AdjustmentNode\*\*:  
 - If triggered by the Decision Agent, adds a new row in the Claims File for adjustments.  
 - Updates:  
 - Copies the `Claim ID` (DCN) and assigns a new `Item Code`.  
 - Updates the `Allowed Amount` and other fields as required.

## Collaboration Between Agents

1. \*\*Data Flow\*\*:  
 - Agents share data through the dependency graph:  
 - Eligibility Agents pass validation results to the Decision Agent.  
 - The Decision Agent conditionally triggers the Adjustment Agent based on the decision output.  
  
2. \*\*Parallel Execution\*\*:  
 - Eligibility Agents execute concurrently via `asyncio.gather()`.  
 - This reduces execution time by eliminating sequential bottlenecks.  
  
3. \*\*Dependency Execution\*\*:  
 - Downstream agents execute only after receiving inputs from their parent nodes.  
 - For example:  
 - The Decision Agent executes only after all Eligibility Agents complete.  
 - The Adjustment Agent executes only if the Decision Agent outputs "Adjust."

## Technical Execution Details

#### \*\*Data Files and Preprocessing\*\*  
- The system uses multiple data sources:  
 - \*\*Membership File\*\*: Member eligibility validation.  
 - \*\*Provider File\*\*: Provider contract validation.  
 - \*\*Provider Network File\*\*: Network status validation.  
 - \*\*Claims File\*\*: Decision-making and adjustment updates.  
- Preprocessing ensures:  
 - Consistent formatting of dates, text, and numbers for cross-file comparison.  
 - Reduced errors in eligibility and decision logic.  
  
#### \*\*Parallelism\*\*  
- \*\*asyncio.gather()\*\* executes:  
 - \*\*MemberEligibilityNode\*\*, \*\*ProviderEligibilityNode\*\*, and \*\*ProviderNetworkNode\*\* concurrently.  
- Results from these agents are aggregated and passed to the Decision Agent for further processing.  
  
#### \*\*GraphChain Dependency Management\*\*  
- \*\*Graph Hierarchy\*\*:  
 - Root: \*\*MemberEligibilityNode\*\* (first agent to start the workflow).  
 - Mid-level: \*\*ProviderEligibilityNode\*\*, \*\*ProviderNetworkNode\*\*, and \*\*DecisionNode\*\*.  
 - End: \*\*AdjustmentNode\*\* (conditionally triggered).  
- The workflow is orchestrated as a directed acyclic graph (DAG) using `add\_parent()` to define dependencies.

## Advantages of This Approach

1. \*\*Scalability\*\*:  
 - New agents can be added by defining additional nodes and connecting them to the existing graph.  
2. \*\*Accuracy\*\*:  
 - Data preprocessing ensures clean and reliable comparisons across multiple files.  
3. \*\*Efficiency\*\*:  
 - Parallel execution of agents reduces workflow latency.  
4. \*\*Modularity\*\*:  
 - Independent agent design ensures maintainability and easier testing.  
5. \*\*Extensibility\*\*:  
 - The framework supports future enhancements, such as LLM integration for summarization.