```
graph LR
  A[User Prompt] --> B(ShieldWrapper: Python API/CLI);
  B --> C{Prompt Preprocessing};
  C --> D(Scanner Engine);
  D -- Matches Rule --> E(Rule Engine);
  E --> F(Decision Router);
  subgraph Action Paths
    F -- Action: Block --> G[Security Log & Alert];
    F -- Action: Transform --> H{Transformed Prompt};
    F -- Action: Log --> G;
    H --> F;
  end
  B -- Passes Original/Transformed --> I(LLM Interface);
  I --> J[LLM Response];
  subgraph Optional Response Evaluation
    J --> K{Response Analysis};
    K -- Semantic Leak/Policy Violation --> G;
  end
  I --> L[Final Output];
  G --> L;
  subgraph Error Handling
    B -- Invalid Rules Path --> M[Error Log & Default Behavior];
    E -- Unreadable Rule File --> M;
    D -- Invalid Rule Pattern --> M;
  end
  M --> L;
```

Revised Logical Path from Input to Final Prompt Execution:

* User Prompt: The user provides an input prompt.
* ShieldWrapper (API/CLI): The application interacts with Shield via its API or CLI, passing the raw prompt and configuration (including the path to the rules file).
* Error Handling (Initial): The ShieldWrapper attempts to load the rules file. If the path is invalid or the file cannot be accessed, an error is logged (Error Log & Default Behavior), and Shield might either fail or proceed with a default safe configuration (if implemented).
* Prompt Preprocessing (Optional): The Preprocessor module applies normalization and basic cleaning to the raw prompt.
* Scanner Engine: The Scanner compares the preprocessed prompt against the loaded rules from the Rule Engine. If a rule has an invalid pattern, an error is logged (Error Log & Default Behavior), and that specific rule might be skipped.
* Rule Engine: Manages the loading and parsing of rules. If a rule file is unreadable (e.g., broken YAML, invalid JSON), an error is logged (Error Log & Default Behavior), and Shield might either fail or load a partial/default rule set.
* Decision Router: This new component receives the matched rule(s) and the current state of the prompt. It determines the sequence and combination of actions to be executed based on the rules. For example, if multiple rules match, the Decision Router ensures that logging happens, transformations are applied in a defined order, and a blocking action, if present, takes precedence.
* Action Paths:
  * Block: If a blocking action is triggered, the Decision Router directs the flow to the Security Log & Alert module, and the prompt is prevented from reaching the LLM.
  * Transform: If a transform action is triggered, the Decision Router sends the prompt to the Transformed Prompt module. The modified prompt is then fed back into the Decision Router to allow for further processing by other matched rules or subsequent transformations.
  * Log: If a logging action is triggered, the Decision Router sends the details to the Security Log & Alert module.

 * ShieldWrapper (Feedback Loop): After all applicable rules have been processed by the Decision Router and any transformations have been applied, the ShieldWrapper receives either the original prompt (if no transformations occurred or only logging actions were taken) or the final Transformed Prompt.

 * LLM Interface: The ShieldWrapper then sends this (potentially transformed) prompt to the configured LLM.

 * LLM Response: The LLM generates a response.

 * Optional Response Evaluation: If activated, the Response Analysis module evaluates the LLM's response.

 * Final Output: The application receives the (potentially filtered) LLM response or a notification that the prompt was blocked.

 * Error Log & Default Behavior: This module handles errors encountered during the rule loading or scanning phases. It logs the errors and can implement fallback mechanisms, such as using a default set of safe rules or preventing any LLM interaction.

Description of Newly Added Modules/Connections:

 * Decision Router: This module acts as a central control point after the Rule Engine identifies matching rules. It orchestrates the execution of associated actions. Its logic ensures that multiple actions (e.g., log, transform, block) are handled correctly, potentially in a specific order, and that the prompt's state is managed as it undergoes transformations.

 * Transform Feedback Loop (H --> F): This explicit connection shows that after a transformation action is applied, the modified prompt (Transformed Prompt) is fed back into the Decision Router. This allows for subsequent rules to act upon the transformed prompt, enabling chained transformations or a blocking rule to be triggered based on the modified content.

 * Error Handling Path (B/E/D --> M): These connections illustrate the framework's robustness in handling potential errors during rule processing. If the ShieldWrapper encounters an invalid rules path, the Rule Engine fails to read a rule file, or the Scanner detects an invalid pattern within a rule, the flow is directed to the Error Log & Default Behavior module. This ensures that errors are logged, and the system can implement graceful fallback strategies to prevent application crashes or unexpected behavior.

This revised architecture provides a clearer and more robust flow, explicitly addressing the handling of multiple actions, the transformation feedback mechanism, and critical error handling for rule processing.