I implemented 2 features.

1. Repo Scanning (Extractor): `src/rules/ai_rules_scan.py`

Where: **`extract_rule_statements_from_markdown(content: str) -> list[str]`**

- On Push/PR, the app scans the repo (the branch you are working, not default branch. the default branch will be scanned only in the case of Landing Page or Analayze Repo), get the file list, find the files matching the patterns like this.

```
AI_RULE_FILE_PATTERNS = [
  "*rules*.md",
  "*guidelines*.md",
  "*prompt*.md",
  "**/*rules*.md",
  "**/*guidelines*.md",
  "**/*prompt*.md",
  ".cursor/rules/*.mdc",
  ".cursor/rules/**/*.mdc",
]

 AI_RULE_KEYWORDS = [
    "Cursor rule:",
    "Claude:",
    "always use",
    "never commit",
    "Copilot",
    "AI assistant",
    "when writing code",
    "when generating",
    "pr title",
    "pr description",
    "pr size",
    "pr approvals",
    "pr reviews",
    "pr comments",
    "pr files",
    "pr commits",
    "pr branches",
    "pr tags",
 ]
```

So what Extractor does :

- Splits content into lines.
- Keeps lines that either:
  - Start with a prefix in `EXTRACTOR_LINE_PREFIXES` (e.g. `"Rule:"`, `"Cursor rule:"`), or
  - Contain a phrase in `EXTRACTOR_PHRASE_MARKERS` (e.g. `"always use"`, `"must have"`).
- Returns a list of statement strings.

`No LLM, no graph, no chain — just one synchronous function.`

2. Agentic Parsing & Translation. (Translator) : `src/rules/ai_rules_scan.py`

Where: `translate_ai_rule_files_to_yaml` and helpers.

Part A – Deterministic (no agent):

- `try_map_statement_to_yaml(statement)`
- Substring match against `STATEMENT_TO_YAML_MAPPINGS`.
- Returns a rule dict or `None`.
- No LLM, no LangGraph.

Part B – When mapping returns `None`:

- Code calls `get_agent("feasibility")` and `agent.execute(rule_description=st)`.
- That agent is the Rule Feasibility Agent in `src/agents/feasibility_agent/` — a LangGraph agent (StateGraph with nodes `analyze_rule_feasibility` and `generate_yaml_config`).
- So the "translator" in the agentic path is this existing Feasibility Agent; there is no separate "Translator Agent" graph.

Flow in one picture

```
Markdown content (e.g. rules.md)
    |
    ▼
extract_rule_statements_from_markdown(content)   ← plain function, no LangGraph
    |
    ▼
list of statements (e.g. ["Do not commit secrets...", ...])
    |
    ▼
for each statement:
    |
    ├─ try_map_statement_to_yaml(statement)  ← plain function, no LangGraph
    |       |
    |       ├─ match → use that rule dict
    |       |
    |       └─ no match ▼
    |
    └─ Feasibility Agent (LangGraph)        ← only agent in this pipeline
            execute(rule_description=statement)
            → analyze_feasibility → generate_yaml_config
            → result.data["yaml_content"]
```