An LLM-Based Agent-Oriented Approach for Automated Code Design Issue Localization

Fraol Batole,

fbatole@tulane.edu

David OBrien

davidob@iastate.edu

Robert Dyer

rdyer@unl.edu

Tien N. Nguyen

tien.n.nguyen@utdallas.edu

Hridesh Rajan

hrajan@tulane.edu

47th International Conference on Software Engineering

Design Issues

- Design debt emerges naturally as systems evolve. E.g., (1) poor modularity, (2) tight coupling, (3) excessive complexity, etc.

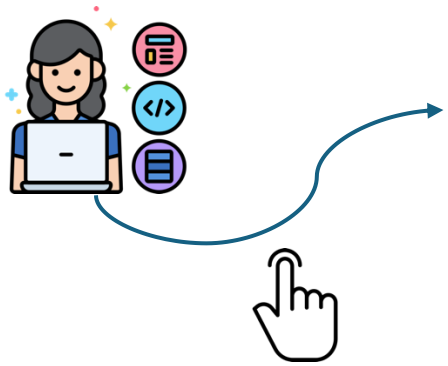
Erode **maintainability** and **drive costs up**.

```
1  ...
2  static void initialiseAndInstallRepository( IConfigurationElement remoteRepositoryElem
       , RepositoryPlugin localRepo,
3    MultiStatus status, IProgressMonitor monitor) {
4   SubMonitor progress = SubMonitor.convert(monitor, 3);
5   - String implicitStr = remoteRepositoryElem.getAttribute("implicit");
6   - String repoName = remoteRepositoryElem.getAttribute("name");
7   - if (repoName == null) repoName = "<unknown>";
8   - if ("true".equalsIgnoreCase(implicitStr))
9     try {
10     - RemoteRepository repo = (RemoteRepository)remoteRepositoryElem.createExecutableExtension("class");
11     repo.initialise(progress.newChild(1));
12     installRepository(repo, localRepo, status, progress.newChild(2));
13    } catch (CoreException e) {
14     String message = MessageFormat.format("Failed to initialise remote repository {0}.", repoName);
15     if (status != null)
16       status.add(new Status(IStatus.ERROR, Plugin.PLUGIN_ID, 0, message, e));
17     Plugin.logError(message, e);
18    }
19    - else {
20      - progress.worked(1);
21    - }
22 } ...
```

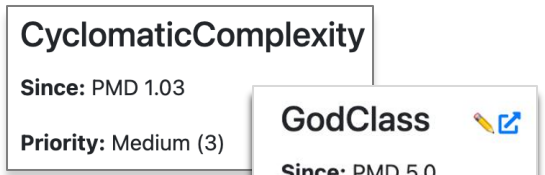
Listing (1) Code Before Refactoring

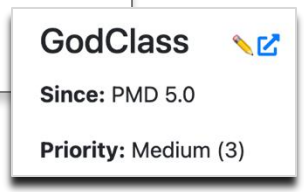
Resolving Design Issues

🔍 Identify the Design Issue



Runs static-analysis tools (e.g., PMD)

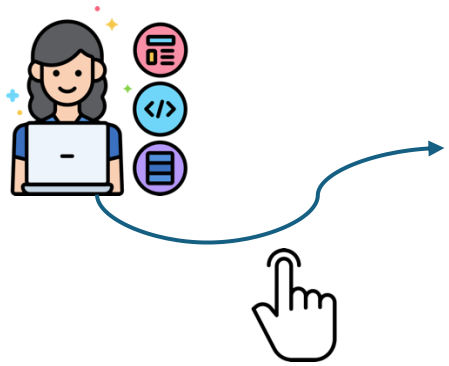




⚠ High-level

Resolving Design Issues

🧠 Understand the Code in Context



Runs more program analysis tools, visual diagrams, read documentation, etc…

- Call Graph
- Program Dependency Graph

Challenges

High-level rule violations

- (*High-Level Detection such as "God Class" ≠ Actionable Localization*)

High cognitive load for developers

Time consuming process

We already have a workflow of resolving the issues from developers

Why LLMs Can't Yet Solve This

Limited context → can't attend to or reason over large connected contexts

LLMs lack built-in understanding of structural artifacts → struggle with PDGs, ASTs, Call graphs, etc.

Need structured inputs to reason effectively

Contributions

Multi-Agent Design Issue Localization Framework

Cooperative agents orchestrating analysis, understand the codebase and localization issues.

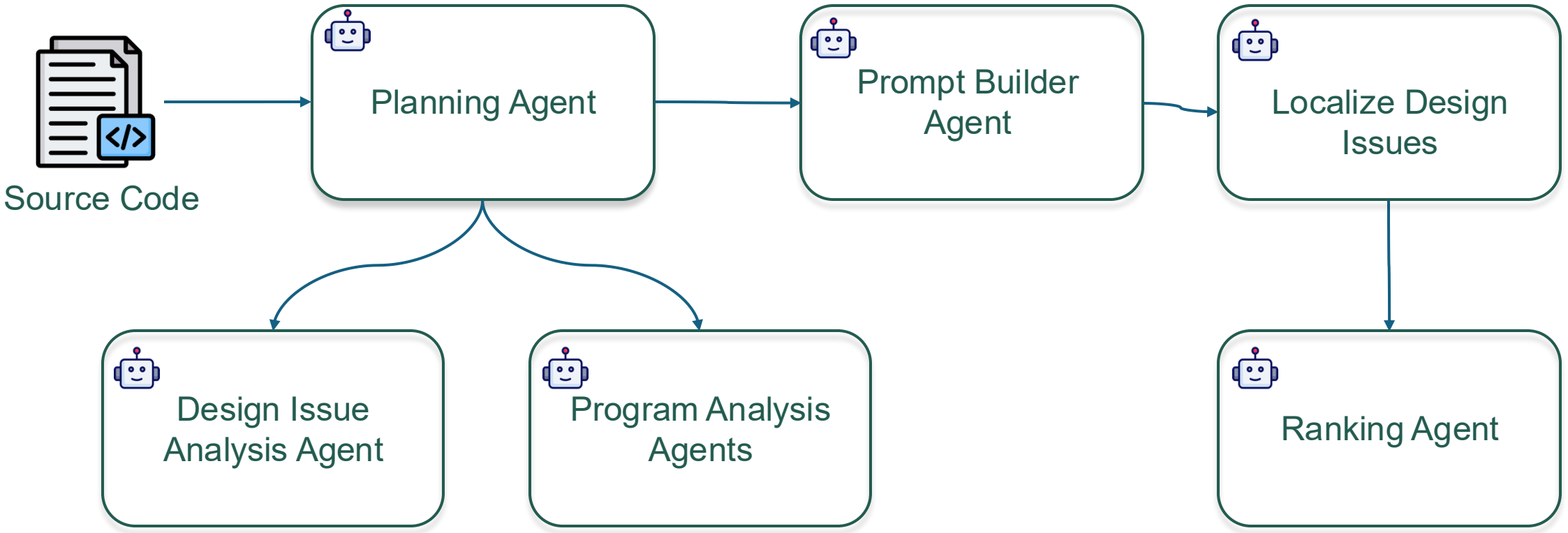
Natural Language Representation of Program Analysis Outputs

Summarized static analysis → LLM-readable insights.

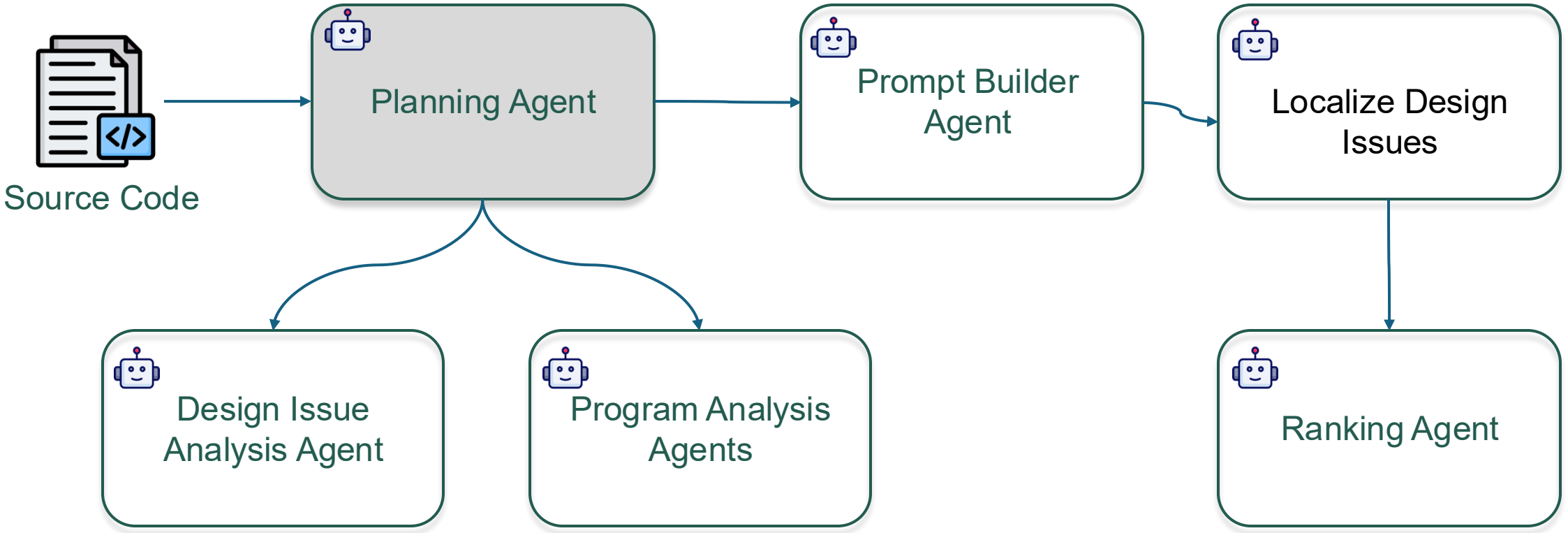
Context-Aware Prompting

Refactoring-type and code analysis-driven prompt customization.

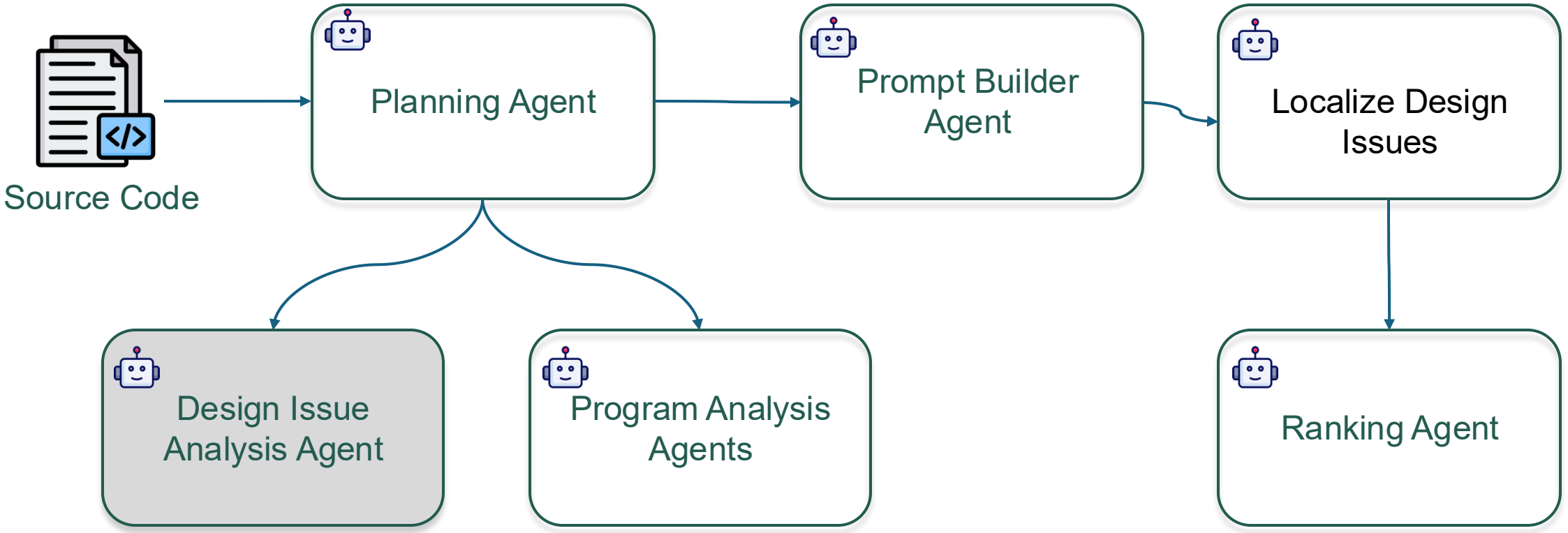
Overview of Approach



Overview of Approach



Overview of Approach



Overview of Approach

Design Issue Analysis Agent

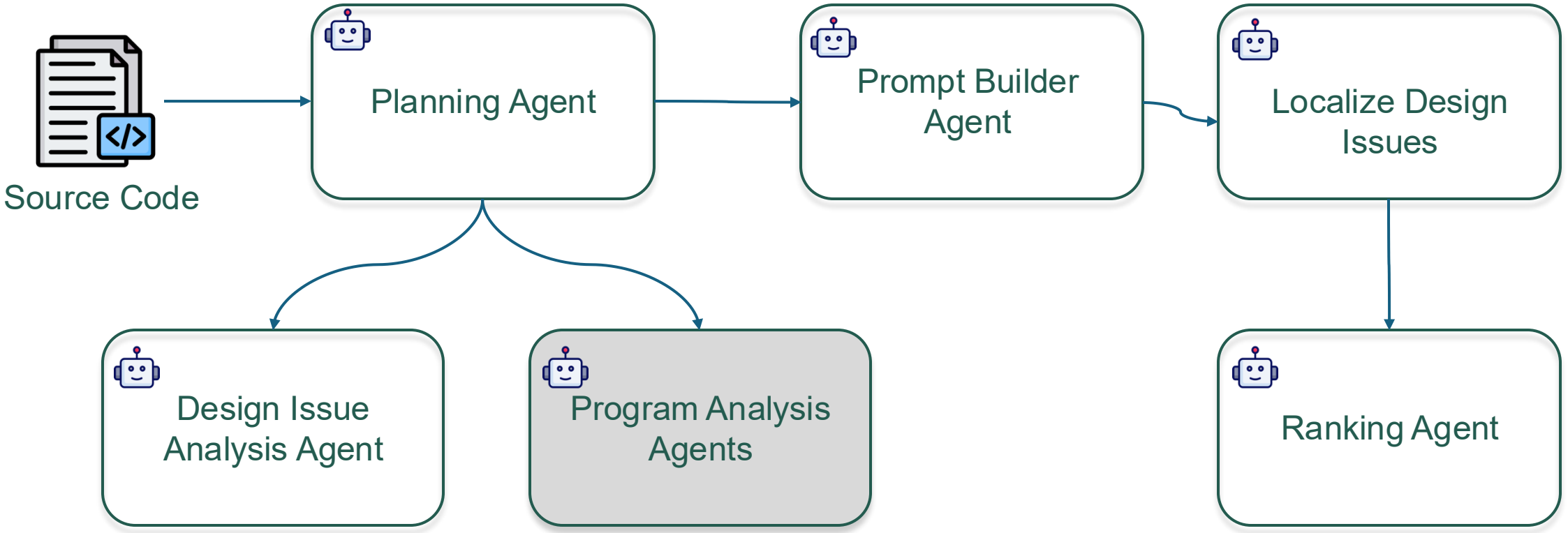
- Uses static-analysis tools (e.g., PMD) to flag coarse-grained smells (God Class, High Coupling, etc.)
- Maps each violation into structural design issue types

God Class
High Coupling
} Modularity

- Passes structured violation summaries to downstream agents (Planning, Program Analysis)

➤ ***Infers the design rationale behind each violation to guide localization planning***

Overview of Approach



Overview of Approach

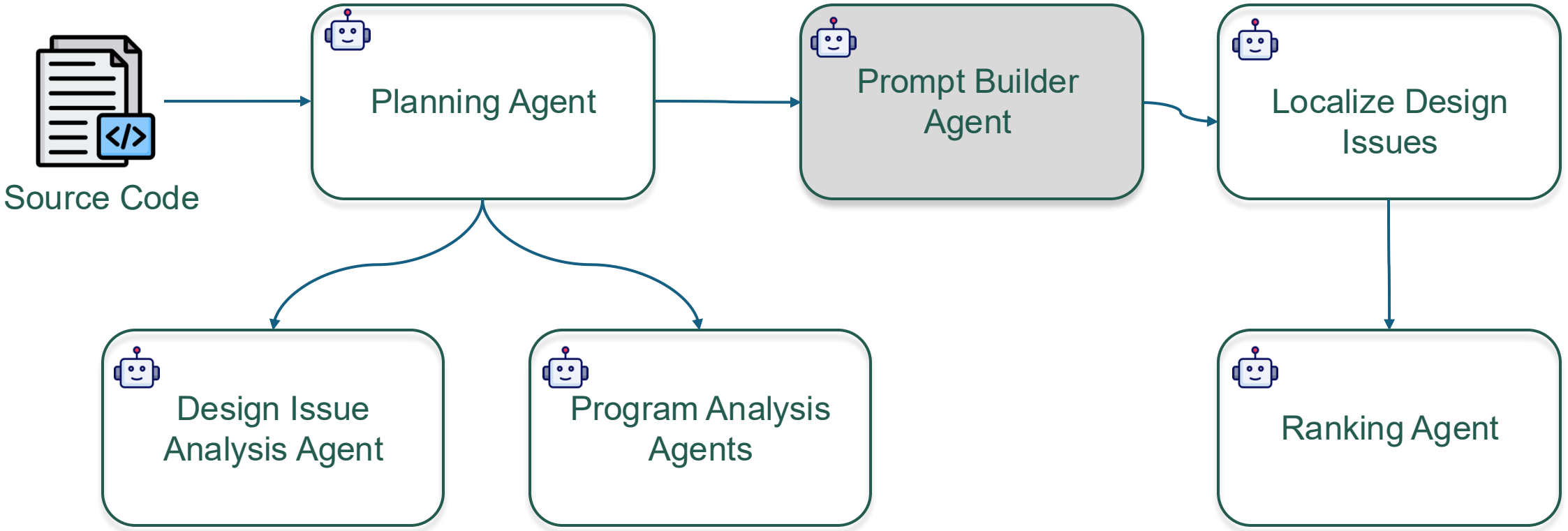
Program Analysis Agents

- **Run lightweight, targeted analyses to extract structural cues from the codebase, e.g.,:**
  - Fan-in / Fan-out metrics
  - Class coupling degrees

- **Provide summaries tailored for LLM consumption**
  - Avoids sending large graphs or raw ASTs

➤ ***Passes compact, LLM-interpretable natural-language summary***

Overview of Approach



Overview of Approach

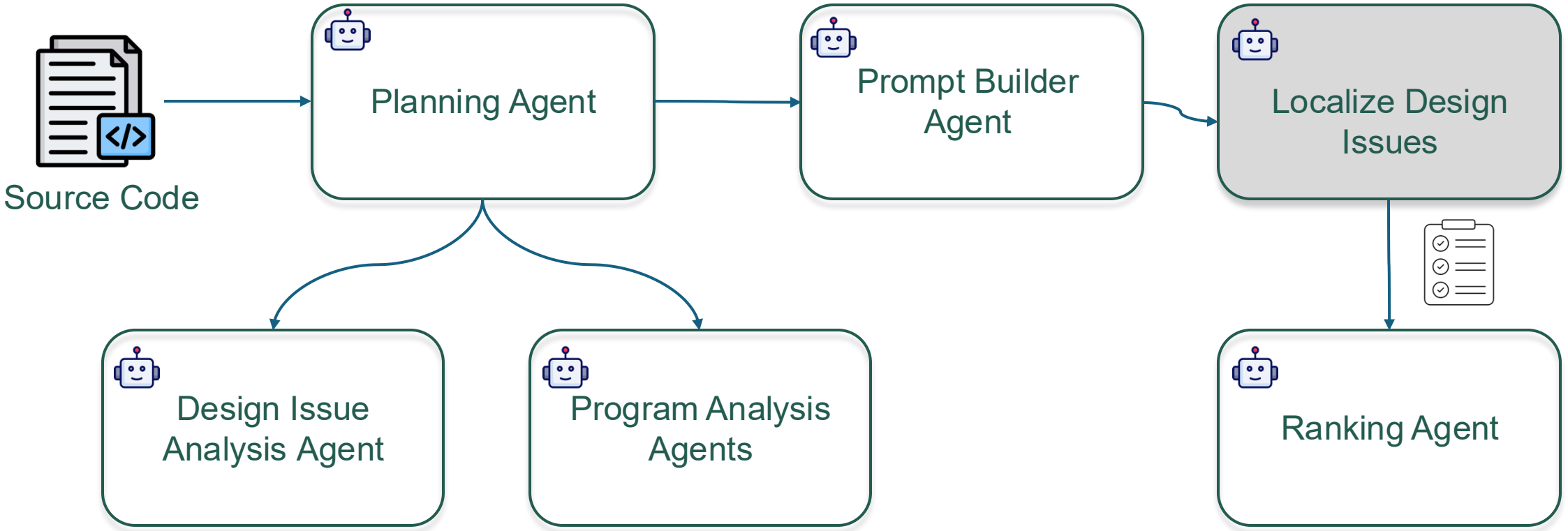
Prompt Builder Agent

- Synthesizes a structured prompt for LLMs from upstream outputs
  - Uses the detected design issue + analysis summaries

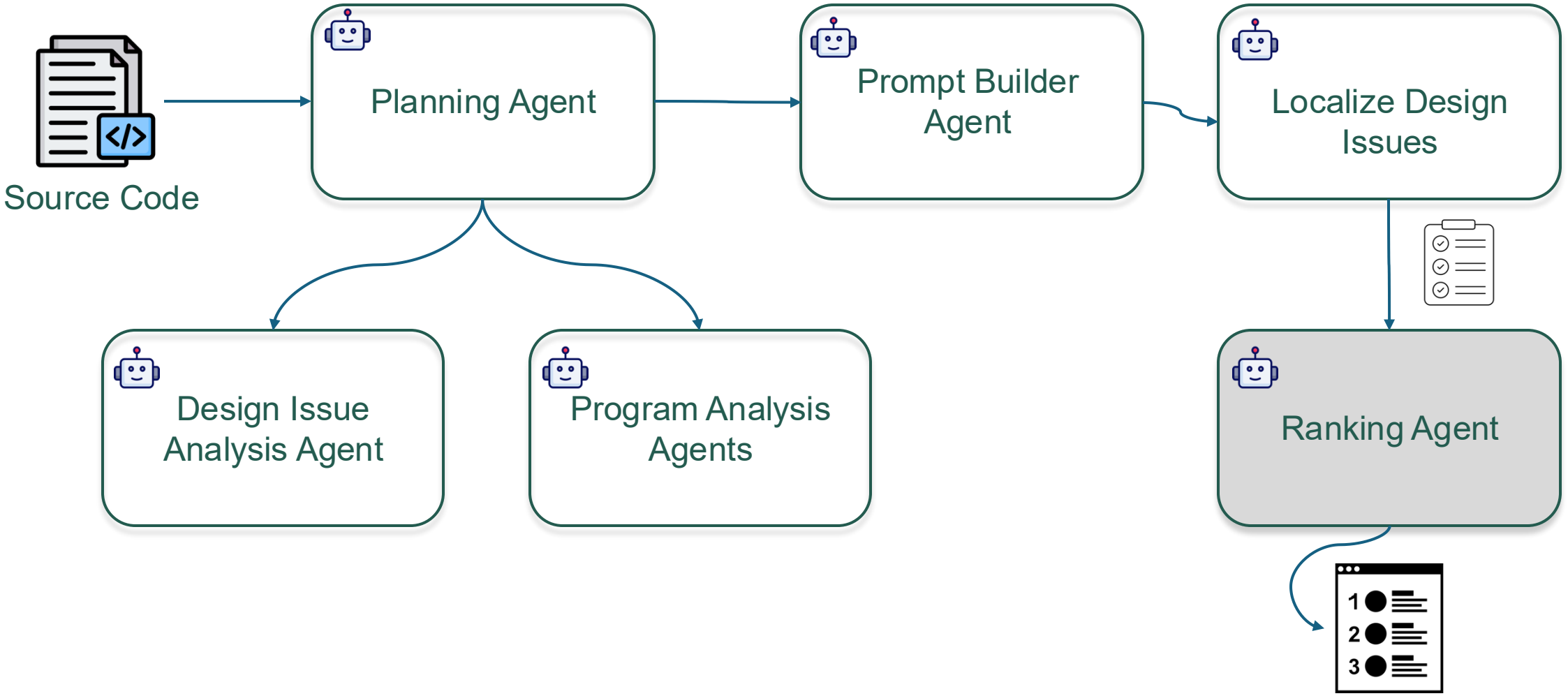
- Fills a 3-part dictionary:
  - **Query** — refactoring intent in question form
  - **Code Snippet** — full relevant code
  - **Analysis Summary** — reasoning over the metrics and structural cues

➤ ***Improves localization precision by grounding the LLM in structural reasoning***

Overview of Approach

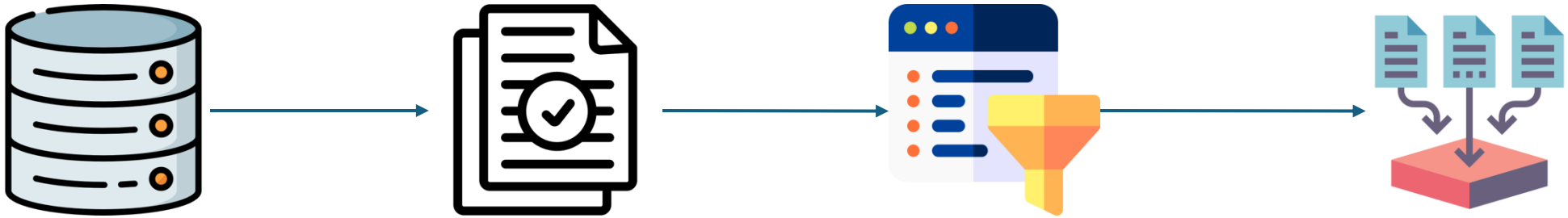


Overview of Approach



Experimental setup

➢ **Dataset Construction Process**



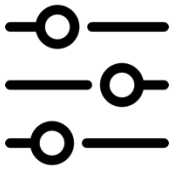
Experimental setup

- **Baseline Comparison:**
  - Comparison against naive LLM prompting method

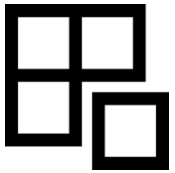
- **Evaluation Metrics:**
  - Exact-match accuracy at varying ranks (EM@1, EM@5, EM@10)

Research Questions

RQ1: How effective is LOCALIZEAGENT on localizing design issues?



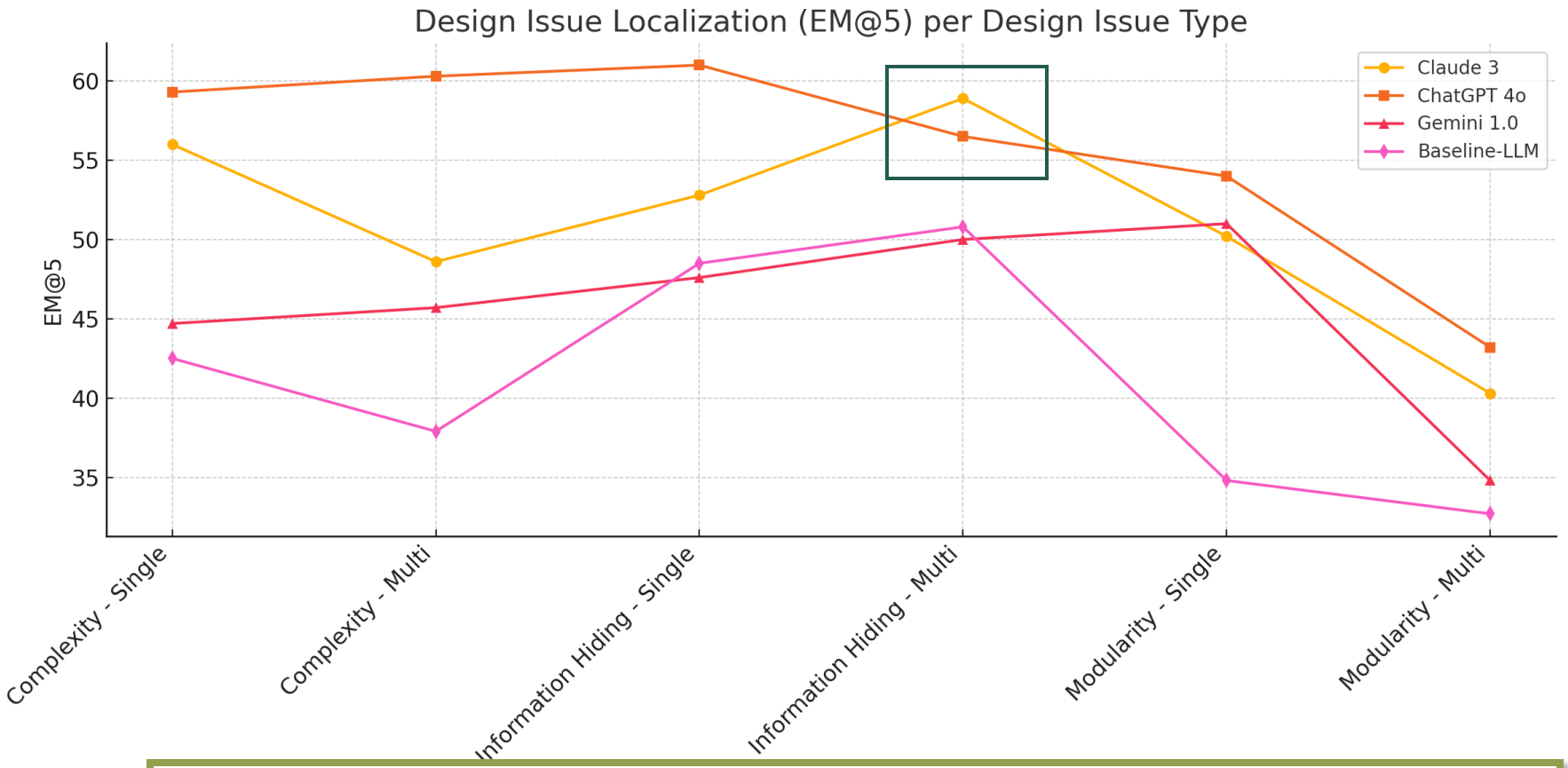
RQ2: How sensitive is LOCALIZEAGENT under different API settings when suggesting design issues refactoring?



RQ3: (Ablation Study) How does each component in LOCALIZEAGENT contribute to the performance of localizing issues?

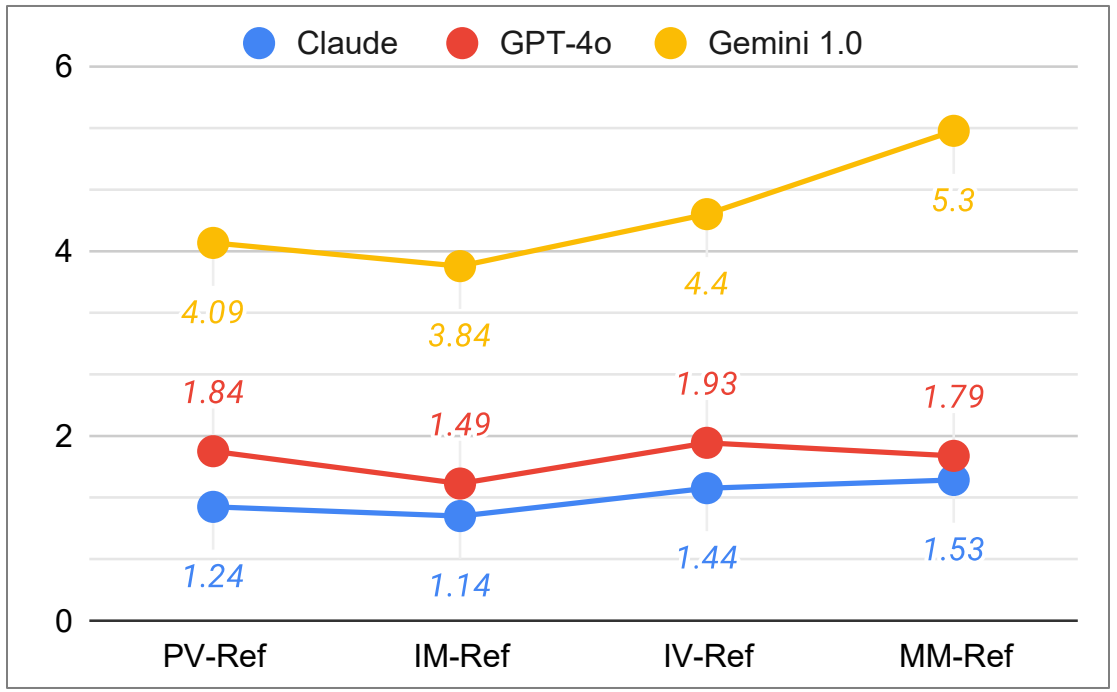
RQ4: What is the time and budget to localize design issues?

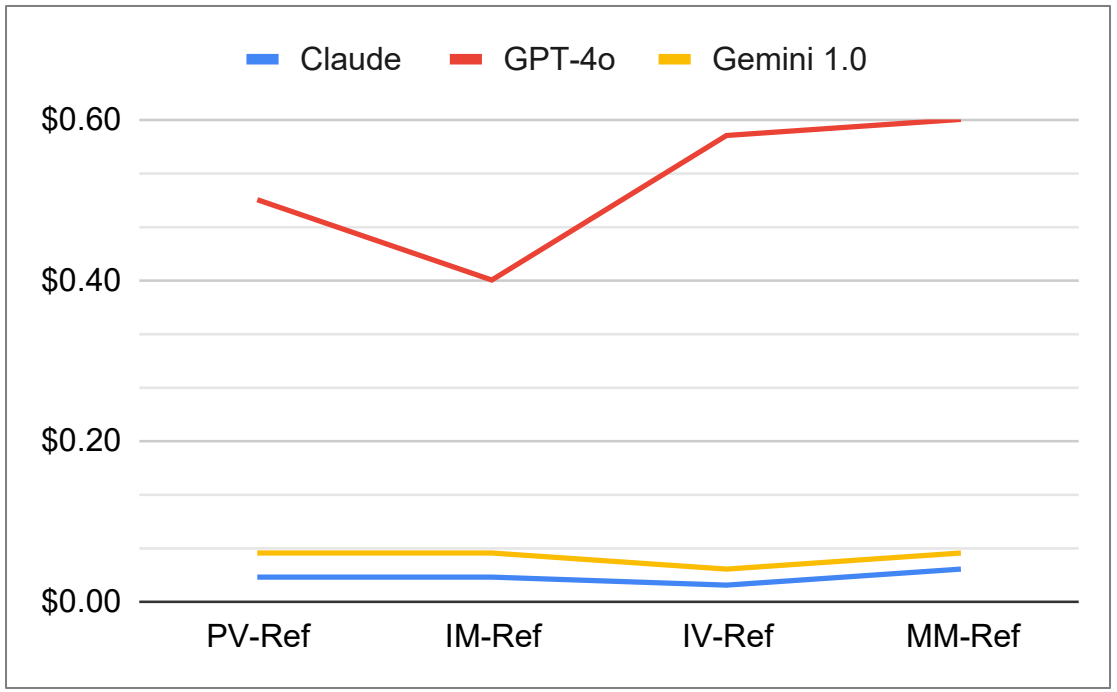
RQ1 Results – Recall



LocalizeAgent significantly improves design issue localization, achieving higher top-k recall than naive prompting, especially for modularity violations.

RQ4 Results – Runtime and Cost





LocalizeAgent offers a cost-effective and time-efficient solution, completing localization tasks with low API usage and minimal latency.

Conclusion

Design Issues

➢Design debt emerges naturally as systems evolve. E.g., (1) poor modularity, (2) tight coupling, (3) excessive complexity, etc.

1 Erode **maintainability** and **drive costs up**.

2

```
1 ...
2  static void initialiseAndInstallRepository( IConfigurationElement remoteRepositoryElem
     , RepositoryPlugin localRepo,
3    MultiStatus status, IProgressMonitor monitor) {
4   SubMonitor progress = SubMonitor.convert(monitor, 3);
5  - String implicitStr = remoteRepositoryElem.getAttribute("implicit");
6  - String repoName = remoteRepositoryElem.getAttribute("name");
7  - if (repoName == null) repoName = "<unknown>";
8  - if ("true".equalsIgnoreCase(implicitStr))
9    try {
10    - RemoteRepository repo = (RemoteRepository)remoteRepositoryElem.createExecutableExtension("class");
11     repo.initialise(progress.newChild(1));
12     installRepository(repo, localRepo, status, progress.newChild(2));
13    } catch (CoreException e) {
14     String message = MessageFormat.format("Failed to initialise remote repository {0}.", repoName);
15     if (status != null)
16      status.add(new Status(IStatus.ERROR, Plugin.PLUGIN_ID, 0, message, e));
17     Plugin.logError(message, e);
18   }
19  - else {
20     - progress.worked(1);
21  - }
22 } ...
```

Listing (1) Code Before Refactoring

Contributions

🤖 **Multi-Agent Design Issue Localization Framework**

Cooperative agents orchestrating analysis and localization.

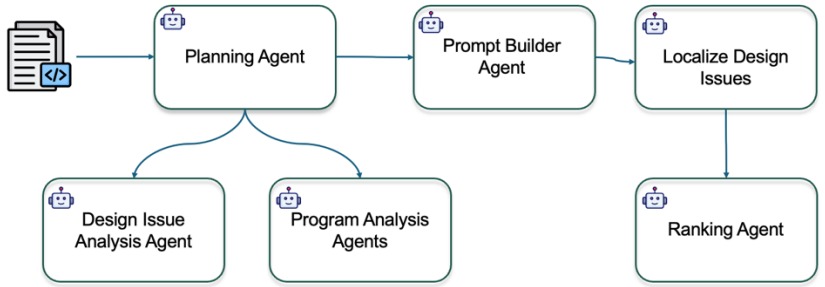
📝 **Natural Language Representation**

Summarized static analysis → LLM-readable insights.

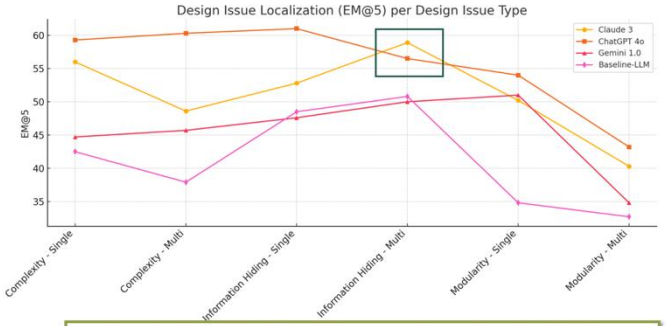
💡 **Context-Aware Prompting:**

Refactoring-type and code analysis-driven prompt customization.

Overview of Approach



RQ1 Results – Recall



LocalizeAgent significantly improves design issue localization, achieving higher top-k recall than naive prompting, especially for modularity violations.

Thank you!

Full Paper:

Codebase:

Please scan the QR code to read the full paper or to access the tool