# Codernize AI – Design Notes

Andrea Paparella

May 18, 2025

## 1 DocGen: Documentation Generation Tool

### 1.1 Introduction

DocGen automates the creation of documentation by scanning a repository for source and configuration files, then generating a single document describing the project.

### 1.2 Pipeline Evolution

Multiple iterations were explored before reaching the current version of the tool that delivers reliable and readable documentation.

**v0 - One File per Doc**

- Walk repository; generate one documentation file per source file.
- *Outcome*: Huge scatter of micro-docs; navigation was impractical.

**v1 - Single Growing Document**

- Append each new file's doc to a single Markdown file as the scan proceeds.
- *Issue*: Recency bias, later files dominated the context window, so earlier sections were forgotten or overwritten.

**v2 - Current Architecture**

1. Scan repository for relevant extensions.
2. Categorise each file.
3. Generate a concise snippet per file (`nano` model).
4. Aggregate all snippets in one shot into a unified draft (`mini` model).
5. Polish the draft with a second `mini`-model pass focused on layout, headings, and flow.

### High-Level Architecture Design

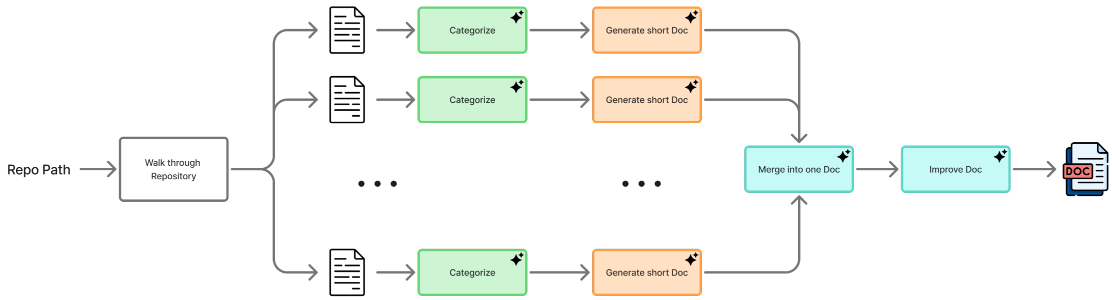

Figure 1: High-level architecture design of DocGen-v2.

### Design Choices: Output Format

Markdown was chosen over AsciiDoc because, despite AsciiDoc's richer syntax, the LLM often generated inconsistent or error-prone output with it. In contrast, Markdown consistently produced well-formatted results.

## 1.3 Model Selection Strategy

| Task | Model | Why |
|---|---|---|
| File categorisation | `gpt-4.1-nano` | Fast, low-latency; prompt is highly constrained (single-word category). |
| Per-file short docs | `gpt-4.1-nano` | Output is short and templated; speed outweighs marginal quality gains of a larger model. |
| Combined summary | `gpt-4o-mini` | Needs larger context window and stronger reasoning to fuse many snippets coherently. |
| Final cleanup / re-org | `gpt-4o-mini` | Same reasons as above; excels at editing and structural refinements. |

## 1.4 Running the Tool

### Set-up

Before running the script, export your OpenAI API key:

```
export OPENAI_API_KEY = <your_openai_api_key>
```

### Usage

```
python3 gen_docs.py /path/to/project -o out_docs -doc docs.md -d
```

- `/path/to/project` – Root directory of the target project.
- `-o out_docs` – Output directory for generated documentation.
- `-doc docs.md` – Name of the final documentation file.
- `-d` – Enables debug mode (detailed logging and per-file documentation snippets).

# 2 ModGen: Modernization Report Generation Tool

## 2.1 Introduction

ModGen generates a modernization report by scanning source and configuration files, identifying modernization opportunities, and suggesting boilerplate code for new architectures (e.g., Spring Boot). To improve understanding of the target repository, the tool also accepts project documentation as input. You can use documentation generated by DocGen for this purpose.

## 2.2 Pipeline Evolution

Fewer iterations were needed due to learnings from DocGen.

### v0 – Current Architecture

1. Scan repository for relevant file extensions.
2. Generate a modernization analysis per file, suggesting Spring Boot migration opportunities.
3. Aggregate the analyses into a unified report, using the repository documentation as additional input to give more comprehensive context to the LLM.

### High-Level Architecture Design

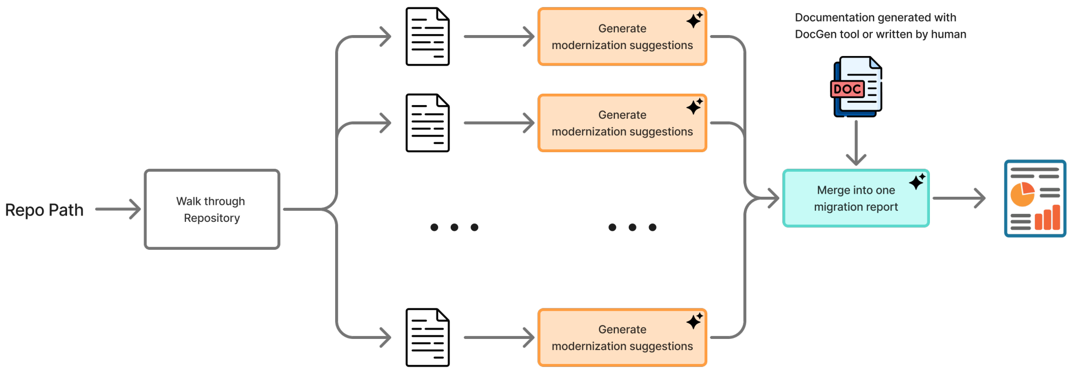
Repo Path
Walk through Repository
Generate modernization suggestions
Generate modernization suggestions
Generate modernization suggestions
Documentation generated with DocGen tool or written by human
DOC
Merge into one migration report

Figure 2: High-level architecture design of ModGen-v0.

### 2.3 Model Selection Strategy

| Task | Model | Justification |
|---|---|---|
| Per-file modernization analysis | `gpt-4.1-mini` | High reasoning capability and large context window to produce detailed analyses and high-quality code. |
| Combined report | `gpt-4.1-mini` | Same reasoning as above; suited for integrating comprehensive insights. |

### 2.4 Running the Tool

#### Setup

Before running, export your OpenAI API key:

```
export OPENAI_API_KEY=<your_openai_api_key>
```

#### Usage

```
python3 mod-gen.py /path/to/docs.md /path/to/project -o out_docs -report
modernization-report.md -d
```

- `/path/to/docs.md` – Documentation path to give the LLM better context.
- `/path/to/project` – Root directory of the target project.
- `-o out_docs` – Output directory for the generated documentation.
- `-report modernization-report.md` – Name of the generated modernization report.
- `-d` – Enables debug mode (detailed logging).

## 3 Future Improvements

- **Extension handling**: The tools currently rely on a hardcoded list of file extensions to identify relevant files (e.g., *.py*, *.js*). This approach is not scalable. One possible improvement is to scan all file extensions in the repository and use an LLM to determine which ones should be included in documentation generation.
- **Handling large files and token limits**: If a file is too large, or if the combined content of multiple files exceeds the model's context window, the API may fail and skip the file. To address this, we could explore tools like LangChain to chunk large files or summarize groups of files before generating documentation. However, this may lead to some information loss. Alternatively, switching to models with larger context windows can help, although they still have upper limits.
- **Parallelization of per-file generation**: Currently, the tool generates documentation or analysis for each file sequentially. This could be optimized by running multiple model calls in parallel, significantly reducing runtime. The aggregation step could then proceed once all per-file snippets are available.