AI-Powered CV Summary (Gemini-Based)

Overview

The AI-Powered CV Summary feature parses a PDF CV, identifies major sections (e.g., Work Experience, Education, Skills), and leverages **Gemini** (an LLM-based summarization service) to generate a concise, human-readable summary of the applicant's background.

---

1. Functional Requirements

ID	Requirement
FR 1.1	The system shall accept a PDF file as input via the command-line interface.
FR 1.2	The system shall extract text from the PDF while preserving basic structure (e.g., headings, bulleted lists) as much as possible.
FR 2.1	The system shall analyze the extracted text to identify major CV sections (e.g., Education, Work Experience, Skills, Contact Info).
FR 2.2	The system may rely on keyword- or heuristic-based detection (e.g., matching "Education," "Experience," etc.) to locate key sections.
FR 3.1	The system shall use the **Gemini LLM** to generate a concise summary of the candidate's background.
FR 3.2	The summary shall emphasize: top skills/areas of expertise, years of experience, highest level of education, and notable achievements.
FR 4.1	The system shall print the final summary to the command line.
FR 4.2	The system may provide an option to export the generated summary to a text file (e.g., summary.txt).
FR 5.1	The user should be able to specify a maximum word/character count for the summary (e.g., --max-words 100).
FR 5.2	The user may provide a specific focus area (e.g., "leadership," "technical skills") to guide the Gemini-based summarization if needed.

---

2. Non-Functional Requirements

ID	Requirement
NFR 1.1	**Performance**: The system should handle a typical 2–4 page CV and produce a summary in under 5 seconds on average hardware.

ID	Requirement
NFR 1.2	**Performance**: Memory usage should remain within reasonable bounds (no excessive in-memory expansions).
NFR 2.1	**Usability**: The command-line interface must be simple to invoke (e.g., python main.py --pdf candidate_cv.pdf --max-words 100).
NFR 2.2	**Usability**: Error messages must be clear and informative (e.g., “File not found,” “Unsupported PDF format”).
NFR 3.1	**Reliability**: The PDF parsing module should handle minor formatting issues or unusual layouts without crashing.
NFR 3.2	**Reliability**: If the system fails to detect key sections, it should still generate a fallback summary (though it may omit missing sections).
NFR 4.1	**Maintainability**: The code shall adhere to PEP 8 (Python style guide).
NFR 4.2	**Maintainability**: The code should be modular and well-documented, with clear separation of concerns (e.g., PDF parsing, Gemini integration, CLI parsing).
NFR 5.1	**Security**: No personal user data (beyond the CV itself) should be stored or transmitted without explicit user consent.
NFR 5.2	**Security**: If using Gemini’s API, credentials or tokens must be handled securely (e.g., environment variables), not hard-coded in the source.

---

3. Acceptance Criteria

ID	Criterion
AC 1.1	**Accuracy & Completeness**: The summary must mention the candidate’s primary field/role, highest educational attainment (if available), and notable skills. It must avoid unrelated or incorrect details.
AC 1.2	**Accuracy & Completeness**: If the candidate has achievements or certifications listed, these should be reflected or referenced in the summary (if space allows).
AC 2.1	**Readability & Style**: The summary must be written in clear, grammatical English.
AC 2.2	**Readability & Style**: The summary must be limited to the maximum length specified (defaulting to 100 words if no limit is provided).
AC 3.1	**Section Coverage**: If sections like Education, Experience, and Skills are detected, they must be referenced in the generated summary. Missing sections should be skipped gracefully.

ID	Criterion
AC 4.1	**CLI Functionality**: Running the CLI with correct flags must produce an output summary within 5 seconds for a 2–4 page CV on typical hardware.
AC 4.2	**CLI Functionality**: Invalid inputs (e.g., missing PDF, unreadable file) should produce an informative error message rather than cause a crash.
AC 5.1	**Test Cases**: At least one automated **unit test** must confirm that the summarizer extracts known sections from a sample CV.
AC 5.2	**Test Cases**: At least one **system-level test** must confirm end-to-end functionality (PDF → Gemini Summarization → CLI output).

How to Use This Document

1. **Implementation Reference**: Use the tables above to guide your PDF parsing, Gemini API integration, and CLI design.
2. **Testing & Validation**: Each acceptance criterion should map to specific test cases to ensure all requirements are met.
3. **Project Milestones**: Track progress by checking off functional and non-functional requirements as you implement and test them.

This table-based format ensures clarity in roles, responsibilities, and success metrics for each requirement tied to the **Gemini-based** summarization feature.

You said:

Write the README Project Phases Each phase must be documented in the README as its own section. The documentation should be included directly in the README (marked below as "inline"), or with links to other parts of the project ("link"). Note that the project phases may not necessarily be executed in the order presented. Phase 1: Requirements Engineering - Consult with an LLM to define ONE significant and interesting CV analysis feature. - Document the feature's requirements clearly (inline). - Include acceptance criteria (inline). - Document LLM interactions (link).

Reasoned for a few seconds

CV Analyzer

Welcome to the **CV Analyzer** project! This command-line application processes a PDF CV, identifies key sections, and generates a concise summary using the **Gemini** LLM. The goal is to make CV screening and review more efficient for both candidates and recruiters.

Table of Contents

1. Project Overview
2. Installation

3. Usage
4. Phase 1: Requirements Engineering

Project Overview

This project is designed to:

- Parse a CV in PDF format.
- Identify major sections (Education, Work Experience, Skills).
- Generate a concise summary using Gemini, focusing on top skills, years of experience, and highest education.

You'll find details about the **design, architecture, and testing** in subsequent phases. For now, we're documenting Phase 1 (Requirements Engineering).

Installation

1. **Clone the Repository** (private):

bash

CopyEdit

```
git clone <your-private-repo-link>.git
```

2. **Create & Activate Conda Environment** (example):

bash

CopyEdit

```
conda create --name CV_Analyzer python=3.9
conda activate CV_Analyzer
```

3. **Install Dependencies**:

bash

CopyEdit

```
pip install -r requirements.txt
```

Make sure you have google-generativeai installed if you plan on using Gemini.

Usage

Below is a **preview** of how to run the tool (detailed usage comes in later phases):

bash

CopyEdit

```
python main.py --pdf path/to/candidate_cv.pdf --max-words 100
```

- --pdf <file> : Path to the candidate's PDF CV.
- --max-words <int> : Limits the summary length (optional).

Output: A concise summary of the CV in your terminal (and optionally saved to summary.txt).

Phase 1: Requirements Engineering

1. Consult with an LLM

We decided on **one significant and interesting feature**: **AI-Powered CV Summary** using the Gemini LLM. Our rationale:

- Advanced text generation for a recruiter-friendly summary.
- Showcases the power of an LLM to handle complex or nuanced CV data.

(See LLM Interactions section for conversation logs.)

2. Feature Requirements (Inline)

ID	Requirement
FR 1.1	The system shall accept a PDF file as input via the command-line interface.
FR 1.2	The system shall extract text from the PDF while preserving basic structure (e.g., headings, bulleted lists) as much as possible.
FR 2.1	The system shall analyze the extracted text to identify major CV sections (e.g., Education, Work Experience, Skills, Contact Info).
FR 2.2	The system may rely on keyword- or heuristic-based detection to locate key sections.
FR 3.1	The system shall use the **Gemini** LLM to generate a concise summary of the candidate's background.
FR 3.2	The summary shall emphasize top skills, years of experience, highest education, and notable achievements.
FR 4.1	The system shall print the final summary to the command line.
FR 4.2	The system may provide an option to export the generated summary to a text file (e.g., summary.txt).
FR 5.1	The user should be able to specify a maximum word/character count for the summary.

ID	Requirement
FR 5.2	The user may provide a specific focus area (e.g., “leadership skills”) to guide the summarization.

Non-Functional Requirements (Excerpt)

ID	Requirement
NFR 1.1	**Performance**: The system should handle a typical 2–4 page CV and produce a summary in under 5 seconds.
NFR 1.2	**Usability**: The CLI must be simple to invoke (e.g., python main.py --pdf cv.pdf --max-words 100).
NFR 1.3	**Security**: If using Gemini, credentials must be stored securely (e.g., .env file).

3. Acceptance Criteria (Inline)

ID	Criterion
AC 1.1	**Accuracy & Completeness**: Summary mentions candidate’s main field, education, and skills.
AC 1.2	**Readability & Style**: Summary must be in clear English; limit to --max-words if provided.
AC 1.3	**CLI Functionality**: Must produce a summary within 5 seconds for a 2–4 page CV on average hardware.
AC 1.4	**Error Handling**: Invalid inputs trigger informative errors (e.g., “File not found”).

4. LLM Interactions (Link)

- Phase 1 LLM Chat Transcript
  A conversation log detailing how we brainstormed and refined the AI-Powered CV Summary feature.

Note: Future phases (Architecture, Design, Coding & Testing, Documentation) will be added to this README with additional links and inline documentation, in line with the project requirements.

Next Steps

- Proceed to **Phase 2: Architecture** where we’ll define the command-line interface, file system interactions, and third-party library dependencies in more detail.
- Continue capturing all relevant LLM conversations in the chats/ directory.

Usage

Below is a **preview** of how to run the tool (detailed usage comes in later phases):

bash

CopyEdit

```
python main.py --pdf path/to/candidate_cv.pdf --max-words 100
```

- --pdf <file> : Path to the candidate's PDF CV.
- --max-words <int> : Limits the summary length (optional).

Output: A concise summary of the CV in your terminal (and optionally saved to summary.txt).

Phase 1: Requirements Engineering

1. Consult with an LLM

We decided on **one significant and interesting feature**: **AI-Powered CV Summary** using the Gemini LLM. Our rationale:

- Advanced text generation for a recruiter-friendly summary.
- Showcases the power of an LLM to handle complex or nuanced CV data.

(See LLM Interactions below for conversation logs.)

2. Feature Requirements (Inline)

ID	Requirement
FR 1.1	The system shall accept a PDF file as input via the command-line interface.
FR 1.2	The system shall extract text from the PDF while preserving basic structure (e.g., headings, bulleted lists) as much as possible.
FR 2.1	The system shall analyze the extracted text to identify major CV sections (e.g., Education, Work Experience, Skills, Contact Info).
FR 2.2	The system may rely on keyword- or heuristic-based detection (e.g., matching "Education," "Experience," "Skills," etc.).
FR 3.1	The system shall use the **Gemini** LLM to generate a concise summary of the candidate's background.
FR 3.2	The summary shall emphasize top skills, years of experience, highest education, and notable achievements or certifications.

ID	Requirement
FR 4.1	The system shall print the final summary to the command line.
FR 4.2	The system may provide an option to export the generated summary to a text file (e.g., summary.txt).
FR 5.1	The user should be able to specify a maximum word or character count for the summary (e.g., --max-words 100).
FR 5.2	The user may provide a specific focus area (e.g., “leadership skills”) to guide the summarization if desired.

Non-Functional Requirements (Excerpt)

ID	Requirement
NFR 1.1	**Performance**: The system should handle a typical 2–4 page CV and produce a summary in under 5 seconds on average hardware.
NFR 1.2	**Usability**: The CLI must be simple to invoke (e.g., python main.py --pdf candidate_cv.pdf --max-words 100).
NFR 1.3	**Security**: If using Gemini’s API, credentials must be stored securely (e.g., in .env, not hard-coded).

3. Acceptance Criteria (Inline)

ID	Criterion
AC 1.1	**Accuracy & Completeness**: The summary must mention the candidate’s main field/role, education, and skills.
AC 1.2	**Readability & Style**: The summary must be written in clear English; limit to --max-words if specified.
AC 1.3	**CLI Functionality**: Must produce a summary within 5 seconds for a 2–4 page CV on average hardware.
AC 1.4	**Error Handling**: Invalid inputs (missing PDF, etc.) must return an informative error message.

4. LLM Interactions (Link)

- Phase 1 LLM Chat Transcript
  Describes how we brainstormed and refined the AI-Powered CV Summary feature using Gemini.

Note: Future phases—**Architecture, Design, Coding & Testing, Documentation**—will appear in updates to this README. Stay tuned!

CopyEdit

You said:

rewrite with the help of this template # CV Analyzer AI Powered CV Summary ## Getting started Welcome to the **CV Analyzer** project! This command-line application processes a PDF CV, identifies key sections, and generates a concise summary using the **Gemini** LLM. The goal is to make CV screening and review more efficient for both candidates and recruiters. --- ## Table of Contents 1. [Project Overview](#project-overview) 2. [Installation](#installation) 3. [Usage](#usage) 4. [Phase 1: Requirements Engineering](#phase-1-requirements-engineering) ## Project Overview This project is designed to: - Parse a CV in PDF format. - Identify major sections (Education, Work Experience, Skills). - Generate a concise summary using Gemini, focusing on top skills, years of experience, and highest education. You'll find details about the **design, architecture, and testing** in subsequent phases. For now, we're documenting Phase 1 (Requirements Engineering). ## Installation

```
cd existing_repo

git remote add origin https://gitlab.com/svision1/cv-analyzer-ai-powered-cv-summary.git

git branch -M main

git push -uf origin main
```

## Add your files ## Integrate with your tools - [ ] [Set up project integrations](https://gitlab.com/svision1/cv-analyzer-ai-powered-cv-summary/-/settings/integrations) ## Collaborate with your team - [ ] [Invite team members and collaborators](https://docs.gitlab.com/ee/user/project/members/) - [ ] [Create a new merge request](https://docs.gitlab.com/ee/user/project/merge_requests/creating_merge_requests.html) - [ ] [Automatically close issues from merge requests](https://docs.gitlab.com/ee/user/project/issues/managing_issues.html#closing-issues-automatically) - [ ] [Enable merge request approvals](https://docs.gitlab.com/ee/user/project/merge_requests/approvals/) - [ ] [Set auto-merge](https://docs.gitlab.com/ee/user/project/merge_requests/merge_when_pipeline_succeeds.html) ## Test and Deploy Use the built-in continuous integration in GitLab. - [ ] [Get started with GitLab CI/CD](https://docs.gitlab.com/ee/ci/quick_start/) - [ ] [Analyze your code for known vulnerabilities with Static Application Security Testing (SAST)](https://docs.gitlab.com/ee/user/application_security/sast/) - [ ] [Deploy to Kubernetes, Amazon EC2, or Amazon ECS using Auto Deploy](https://docs.gitlab.com/ee/topics/autodevops/requirements.html) - [ ] [Use pull-based deployments for improved Kubernetes management](https://docs.gitlab.com/ee/user/clusters/agent/) - [ ] [Set up protected environments](https://docs.gitlab.com/ee/ci/environments/protected_environments.html) *** # Editing this README When you're ready to make this README your own, just edit this file and use the handy template below (or feel free to structure it however you want - this is just a starting point!). Thanks to [makeareadme.com](https://www.makeareadme.com/) for this template. ## Suggestions for a good README Every project is different, so consider which of these sections apply to yours. The sections used in the template are suggestions for most open source projects. Also keep in mind that while a README can be too long and detailed, too long is better than too short. If you think your README is too long, consider utilizing another form of documentation rather than cutting out information. ## Name Choose a self-explaining name for your project. ## Description Let people know what your project can do specifically. Provide

context and add a link to any reference visitors might be unfamiliar with. A list of Features or a Background subsection can also be added here. If there are alternatives to your project, this is a good place to list differentiating factors. ## Badges On some READMEs, you may see small images that convey metadata, such as whether or not all the tests are passing for the project. You can use Shields to add some to your README. Many services also have instructions for adding a badge. ## Visuals Depending on what you are making, it can be a good idea to include screenshots or even a video (you'll frequently see GIFs rather than actual videos). Tools like ttygif can help, but check out Asciinema for a more sophisticated method. ## Installation Within a particular ecosystem, there may be a common way of installing things, such as using Yarn, NuGet, or Homebrew. However, consider the possibility that whoever is reading your README is a novice and would like more guidance. Listing specific steps helps remove ambiguity and gets people to using your project as quickly as possible. If it only runs in a specific context like a particular programming language version or operating system or has dependencies that have to be installed manually, also add a Requirements subsection. ## Usage Use examples liberally, and show the expected output if you can. It's helpful to have inline the smallest example of usage that you can demonstrate, while providing links to more sophisticated examples if they are too long to reasonably include in the README. ## Support Tell people where they can go to for help. It can be any combination of an issue tracker, a chat room, an email address, etc. ## Roadmap If you have ideas for releases in the future, it is a good idea to list them in the README. ## Contributing State if you are open to contributions and what your requirements are for accepting them. For people who want to make changes to your project, it's helpful to have some documentation on how to get started. Perhaps there is a script that they should run or some environment variables that they need to set. Make these steps explicit. These instructions could also be useful to your future self. You can also document commands to lint the code or run tests. These steps help to ensure high code quality and reduce the likelihood that the changes inadvertently break something. Having instructions for running tests is especially helpful if it requires external setup, such as starting a Selenium server for testing in a browser. ## Authors and acknowledgment Show your appreciation to those who have contributed to the project. ## License For open source projects, say how it is licensed. ## Project status If you have run out of energy or time for your project, put a note at the top of the README saying that development has slowed down or stopped completely. Someone may choose to fork your project or volunteer to step in as a maintainer or owner, allowing your project to keep going. You can also make an explicit request for maintainers.

Reasoned for 5 seconds

md

CopyEdit

# CV Analyzer – AI-Powered CV Summary

Welcome to the **CV Analyzer** project! This command-line application processes a PDF CV, identifies key sections (Education, Work Experience, Skills), and generates a concise summary using the **Gemini** LLM. The goal is to make CV screening and review more efficient for both candidates and recruiters.

## Getting Started

1. **Clone the Repository (GitLab)**

```bash
cd existing_repo
git remote add origin https://gitlab.com/svision1/cv-analyzer-ai-powered-cv-summary.git
git branch -M main
git push -uf origin main
```

2. **Add Your Files**
    - Place your Python scripts (including main.py, etc.) in this repo.
    - Commit and push changes to keep your code tracked in GitLab.
3. **Integrate with Your Tools**
    - Set up project integrations (CI/CD, Slack notifications, etc.)
4. **Collaborate with Your Team**
    - Invite team members and collaborators
    - Create a new merge request
    - Automatically close issues from merge requests
    - Enable merge request approvals
    - Set auto-merge
5. **Test and Deploy**
    - Get started with GitLab CI/CD
    - Enable Static Application Security Testing (SAST)
    - Deploy to Kubernetes, Amazon EC2, or Amazon ECS
    - Use pull-based deployments for Kubernetes management
    - Set up protected environments

Table of Contents

1. Project Overview
2. Installation

3. Usage
4. Phase 1: Requirements Engineering
5. Support
6. Roadmap
7. Contributing
8. Authors and Acknowledgments
9. License
10. Project Status

Project Overview

This project is designed to:

- Parse a CV in PDF format.
- Identify major sections (Education, Work Experience, Skills).
- Generate a concise summary using **Gemini**, focusing on top skills, years of experience, and highest education.

Why Gemini?

- Gemini (Google Generative AI) provides advanced LLM capabilities, enabling more human-like text summarization and context understanding than classical NLP libraries.

Key Features:

- Command-line interface for easy automation.
- Extensible architecture (e.g., add new analysis features in future phases).
- Secure handling of credentials (for Gemini API).

Installation

Below is a generic outline for setting up your local environment. Adjust as needed for your system.

1. **Clone the Repository**

bash

CopyEdit

```
git clone https://gitlab.com/svision1/cv-analyzer-ai-powered-cv-summary.git
```

2. **Create & Activate Conda Environment** (example):

bash

CopyEdit

```
conda create --name CV_Analyzer python=3.9
```

```
conda activate CV_Analyzer
```

3. **Install Dependencies**

bash

CopyEdit

```
pip install -r requirements.txt
```

Make sure google-generativeai or any other necessary packages are listed in requirements.txt.

Usage

1. **Basic Command**

bash

CopyEdit

```
python main.py --pdf path/to/candidate_cv.pdf --max-words 100
```

   - --pdf <file>: Path to the candidate’s PDF CV.
   - --max-words <int>: Limits the summary length (optional).

2. **Example Output**
   - A concise summary displayed in your terminal.
   - (Optional) Saved to summary.txt if implemented in future versions.

Phase 1: Requirements Engineering

Overview

For the **AI-Powered CV Summary**, we focus on using **Gemini** to generate a human-readable summary.

1. Consult with an LLM

- We refined our feature with Gemini-based brainstorming (see LLM transcripts in ./chats/phase1_llm_chat.txt).

2. Requirements (Inline)

ID	Requirement
FR 1.1	The system shall accept a PDF file as input via the command-line interface.

ID	Requirement
FR 1.2	The system shall extract text from the PDF while preserving basic structure (headings, bulleted lists).
FR 2.1	The system shall detect major CV sections (Education, Work Experience, Skills, Contact Info).
FR 3.1	The system shall use **Gemini** to summarize the candidate's background.
FR 3.2	The summary shall emphasize top skills, years of experience, highest education, and notable achievements.
FR 4.1	The user can specify output length (e.g., --max-words) and a focus area (e.g., "leadership skills").

Non-Functional Requirements (Excerpt)

ID	Requirement
NFR 1.1	Performance: Should handle a 2–4 page CV and produce a summary in under 5 seconds on average hardware.
NFR 1.2	Usability: A simple command-line interface (main.py --pdf cv.pdf) is required.
NFR 1.3	Security: Gemini API credentials must be stored securely (e.g., .env), not hard-coded.

3. Acceptance Criteria (Inline)

ID	Criterion
AC 1.1	Accuracy & Completeness: Must mention the candidate's role, education, and top skills.
AC 1.2	Readability & Style: Limit summary to --max-words (default 100) in clear English.
AC 1.3	CLI Functionality: Must produce a summary within 5 seconds for a 2–4 page CV.
AC 1.4	Error Handling: Invalid inputs (missing PDF) must show an informative error message.

For more detail, see: LLM Interactions

Support

For questions or issues:

- **Open an issue** in the GitLab repository, or
- Email the maintainers (see Authors and Acknowledgments).

Roadmap

- **Phase 2: Architecture** (Inline/Link)
    - Define CLI specs, file system interactions, and third-party libraries in detail.
- **Phase 3: Design** (Inline/Link)
    - CRC design for classes and modules.
- **Phase 4: Coding & Testing** (Inline/Link)
    - Implement the final code, unit tests, system-level functional test.
- **Phase 5: Documentation** (Inline/Link)
    - Expand README with final usage examples, future enhancements.

Contributing

1. **Fork** or **Clone** the repo.
2. **Create** a feature branch: git checkout -b feature/new-analysis.
3. **Commit** your changes and push.
4. **Create** a Merge Request in GitLab to have your changes reviewed.

Please ensure your code adheres to **PEP 8** and includes appropriate tests.

Authors and Acknowledgments

- **Project Maintainer**: [Your Name / Team Name]
- **Instructor**: [Instructor's handle or name]
- **Contributors**: Special thanks to classmates who reviewed or contributed ideas.

License

If this project is open source, add your chosen license here. Otherwise, specify any usage restrictions.

Project Status

Ongoing (Phase 1 complete)

- We have clarified requirements and acceptance criteria.
- **Next Up**: Move to architecture design (Phase 2) and incorporate feedback from LLM transcripts and instructor guidance.

Thanks to makeareadme.com for the template suggestions.

CopyEdit

You said:

Phase 2: Architecture

Reasoned for 4 seconds

md

CopyEdit

```md
## Phase 2: Architecture


In this phase, we define how the system is structured and how the user interacts with it. We cover the **command-line interface**, **file system interactions**, **third-party libraries**, and **team responsibilities**. We also include references to the relevant LLM interaction logs.


---


### 1. Command-Line Interface Specification


| **CLI Option**        | **Description**                                                                    | **Example**                                    |
|-------------------------|-----------------------------------------------------------------------------------|------------------------------------------------|
| `--pdf <file_path>`      | **Required.** Path to the PDF CV you want to analyze.                                   | `python main.py --pdf my_resume.pdf`           |
| `--max-words <integer>`  | **Optional.** Caps the word count of the Gemini-generated summary (default 100 words). | `python main.py --pdf my_resume.pdf --max-words 80` |
| `--focus <keyword>`      | **Optional.** Guides Gemini to emphasize specific topics (e.g., “leadership”).       | `python main.py --pdf my_resume.pdf --focus leadership` |


**Flow**:
1. **Parse CLI arguments** (validate PDF path, set defaults if optional flags aren’t provided).
```

2. **Extract text** from the PDF.

3. **Identify sections** (heuristic or keyword-based).

4. **Send text to Gemini** (with optional focus, word-limit constraints).

5. **Print** summary to console (and optionally save to file).

### 2. File System Interactions

1. **Input**

   - **PDF File**: Read from user-specified path (`--pdf`).

   - **Credentials**: Load Gemini API key from a **`.env`** file or environment variable to avoid hard-coding sensitive info.

2. **Output**

   - **Terminal Output**: Print the CV summary directly to the command line.

   - **Optional Text File**: If specified in future enhancements, write the summary to something like `summary.txt`.

Directory Structure (Proposed):

CV_Analyzer/ ├── main.py # Entry point for CLI ├── requirements.txt # Dependencies ├── README.md # Documentation ├── .env # Gemini API key (not committed) ├── src/ │ ├── pdf_parser.py # PDF parsing utilities │ ├── summarizer.py # Gemini LLM integration │ └── cli.py # Argument parsing └── chats/ └── phase2_llm_chat.txt # LLM logs for Architecture decisions

yaml

CopyEdit

### 3. Third-Party Libraries

| **Library** | **Purpose** | **Notes** |

|-----------------------|--|--|
`PyPDF2` (or similar)	Extract text from PDFs	Allows reading multi-page PDFs, though advanced formatting may require additional solutions.
`google-generativeai`	Interface with Gemini LLM for text generation	Must securely configure API key.
`python-dotenv`	Manage environment variables (.env)	Helps load Gemini credentials at runtime.
`argparse` or `click`	Parse command-line arguments	Provides user-friendly CLI.

If additional Python libraries are needed for logging, error handling, or testing (e.g., `pytest`), they should be listed in `requirements.txt`.

### 4. Team Member Responsibilities

Member	**Main Responsibilities**
Alice	*CLI & Arg Parsing, Setting up environment*
Bob	*PDF Parsing logic, PDF text extraction*
Carol	*Gemini Summarization integration, Prompt engineering*
Dave	*Testing and QA (unit tests, system-level tests, CI/CD setup)*

(Adjust as needed based on your actual team.)

### 5. LLM Interactions

For detailed discussions and prompt engineering choices regarding this architecture:

- [Phase 2 LLM Chat Transcript](./chats/phase2_llm_chat.txt)

(Create or update this ` .txt` file with your relevant LLM interactions.)

Next Steps

Now that the architecture is defined, we’ll proceed to **Phase 3: Design**, where we’ll refine class res