ChatGPT

Execution Plan for AI-Powered Resume Personalization Tool

This plan outlines how to build a Python command-line tool that automates tailoring a LaTeX resume for a specific job. By combining LaTeX's consistent formatting with AI-driven content suggestions, the tool will quickly produce a customized resume for a given Job Description (JD). LaTeX ensures a polished, easily modifiable template for each job application linkedin.com , and an AI agent (ChatGPT or Google Gemini) will analyze the JD to extract key requirements and suggest targeted changes (ensuring relevant keywords are included for ATS optimization) linkedin.com . The process is divided into sequential phases, each with detailed technical steps. Finally, we outline how to extend the CLI tool into a minimal web application.

Phase 1: Initial Setup and Input Handling

1. **Command-Line Interface & Arguments** – Set up the CLI using Python's `argparse` (or a similar library) to accept input parameters:
    - **Job Description (JD)**: allow input via a file path (to a `.txt` or `.pdf` containing the JD) or direct text input. If a PDF or DOCX is provided, include a preprocessing step to extract text (using libraries like `PyMuPDF` or `python-docx`).
    - **Role Name and Company Name**: accept these as arguments (e.g. `--role "Software Engineer" --company "TechCorp"`).
    - **AI Agent Choice**: an option (e.g. `--agent`) for the user to specify which AI to use (`chatgpt` by default, or `gemini`). This will determine which API the tool calls in Phase 2.

2. **Base LaTeX Template Loading** – Load the user-provided base resume template (a `.tex` file). This template contains all possible content (projects, skills, etc.), where some projects/sections are initially commented out:
    - Read the `.tex` file into memory as a list of lines or a single text blob.
    - Parse the content to identify **optional sections** (e.g. projects wrapped in comment markers `%`). For example, store each project entry (or section) in a structure indicating whether it's active or commented out.
    - Ensure the template has a clear structure (skills section, projects section, etc.), possibly using consistent markers (like section headings or LaTeX comments) that the script can locate.
3. **Output Directory Preparation** – Determine the output folder path based on Company and Role:
    - Construct a path `<Company_Name>/<Role_Name>/` (e.g. `TechCorp/Software Engineer/`). If it doesn't exist, create the directories.
    - If a previous resume file exists in this location (e.g. `resume.tex` or `resume.pdf` from an earlier run), note that for Phase 3 (learning from history). Otherwise, plan to save new outputs here.
    - This directory will be used to store the modified `.tex` and the compiled PDF.

Phase 2: AI Agent Integration and JD Analysis

1. **AI Agent Selection** – Initialize the chosen AI model's API:
    - If the user selected ChatGPT, prepare to use the OpenAI API (e.g. `openai.ChatCompletion` with a GPT-4 model). If Gemini is selected, prepare calls to the Google Gemini API. The tool should handle authentication (e.g. OpenAI API key or Google credentials) via environment variables or a config file.
    - Provide a configuration flag or environment setting that makes switching models easy. For example, one could set an environment variable to use Google's Gemini API as a free alternative to ChatGPT github.com github.com . The code will check this flag and route requests to the appropriate API client.

2. **Prompt Construction (JD Analysis)** – Build a prompt to feed the job description and context to the AI:
    - Include the **Role and Company** in the prompt to focus the context (e.g. *"Analyze the following job description for the **[Role]** position at **[Company]**…"*).
    - Provide the full **Job Description text** for analysis. Instruct the AI to extract the key responsibilities, skills, and keywords sought by the employer (the AI will essentially perform NLP on the JD to find what's important linkedin.com).
    - Add guidance for the output format. For clarity and easy parsing, ask the AI to respond in a structured format (for example, JSON or a bullet list) containing specific fields:
        - A list of which **projects/experience** from the resume should be **included or emphasized** given the JD.
        - Any projects/skills that are **not relevant and can be de-emphasized or removed**.
        - Important **keywords or skills** from the JD that should be inserted into the resume text (especially if currently missing).
        - Suggestions for **new content** (e.g. a new project or bullet point) if the JD calls for something not covered in the base resume.
    - Emphasize that the AI **must not fabricate qualifications**. For example: *"Only use the candidate's actual experience; do not make up information."* This ensures the output remains truthful linkedin.com .
3. **AI Request and Response** – Send the prompt to the AI API and receive the response:
    - For ChatGPT (OpenAI), use a chat completion call with the system/user message paradigm. For Gemini, use its corresponding API (ensuring the request payload matches their requirements).
    - Parse the AI's response. If structured as JSON or a list, convert it into a Python dict or list for easy access. If the response is free-form text, the tool may need to parse it (e.g. look for keywords like "Include:" or "New bullet:").

- Extract from the response the actionable items: which projects to uncomment or remove, which keywords to add, and any new project description text. For example, the AI might respond that certain projects should be highlighted and say *"Include Project Alpha (experience with cloud services), exclude Project Beta (not relevant), add keywords: Kubernetes, CI/CD"*. The tool will interpret these instructions for the next phase.
- (Optional) If the AI's response is not sufficiently structured or misses information, the tool can apply a second prompt. For instance, if the AI indicates a new project is needed but doesn't provide text, ask: *"Please draft a 2-3 sentence project description showcasing [skill] in the context of [Role]."* This ensures we have content to insert for new sections.

Phase 3: Incorporating Context from Past Resumes (Learning from History)

1. **Retrieve Previous Resume Data** – Using the Company/Role inputs, look for an existing resume in the structured folder system:
   - Check if `<Company_Name>/<Role_Name>/resume.tex` (or `resume.pdf`) exists from a prior run. If so, this is a previously tailored resume for the same target. Also consider checking similar roles or same company in case there's relevant prior work (e.g. if applying to a new role at the same company, the old resume for that company might still provide useful context).
   - If a previous `.tex` is found, open and read its content. If only a PDF exists, use a PDF-to-text tool to extract the text content for analysis.
   - Prepare the previous resume content for comparison by stripping out irrelevant parts (like generic header, personal info) so we focus on projects and experience sections.
2. **Analyze Differences from Base Template** – If a prior tailored resume is available, compute how it differed from the base template:
   - Compare the previous resume's content to the base template (this could be a simple diff or a semantic comparison). Identify which projects were uncommented or added in that version and which were left out. Note any keywords that were inserted that are not in the base template.

- Summarize the changes as a quick reference (e.g. “Last time for this role/company, the following projects were included: X, Y, and skill keywords A, B, C were added.”).
- This summary can inform the AI or the tool’s logic in the current run. For instance, if the same role was tailored before, the tool might prioritize similar sections this time.

3. **Provide Historical Context to AI (if applicable)** – Incorporate the past resume info into the AI’s decision-making:
   - If a previous resume is relevant, include it in the AI prompt as additional context. For example: *“Here is the candidate’s resume used previously for a similar application:\n[Previous resume text]\nUsing this as reference, ...”*. This can guide the AI to maintain consistency with what worked before linkedin.com .
   - Alternatively, explicitly tell the AI which projects were used last time and ask if those are suitable for the new JD. For example: *“In a past application for a similar role, projects X and Y were highlighted. Given the new JD, should we use the same? Are there others more relevant?”* The AI can then factor this in its suggestions.
   - Ensure the AI still focuses on the new JD’s specifics, as different companies may value different things. The historical data is a guide, not a strict template.
4. **Machine Learning Consideration** – (Optional advanced step) Over time, accumulate data from multiple resumes and JDs to improve the tool’s suggestions:
   - Maintain a small database (or even just a JSON log) of past JD-to-resume decisions. The tool could use simple heuristics or ML to learn which keywords or projects tend to be chosen for certain roles.
   - For example, if “Project Alpha” was consistently selected for all cloud engineering roles, the tool could automatically prioritize it when a new cloud-related JD is processed.
   - This could be further enhanced by computing embeddings of JDs and past resumes to find the most similar past scenario for a new JD, but this is an extension beyond the initial scope.

Phase 4: Automated Resume Modification (Applying AI Suggestions to LaTeX)

1. **Uncomment or Swap Relevant Projects** – Modify the LaTeX template based on AI's inclusion/exclusion recommendations:
    - For each project or section that the AI marked to **include** (or keep), ensure it is uncommented in the `.tex` file. This may involve removing `%` comment markers at the start of those lines. If an entire block (multiple lines) is commented out for a project, remove the comment symbol for all those lines.
    - For sections the AI suggested to **exclude**, do the opposite: comment them out (e.g. by prefixing lines with `%`). This could mean commenting an entire `\headedsubsection{...}` block or similar. Keep the original text but make it inactive in the LaTeX source.
    - If the AI suggested reordering projects (e.g. putting a more relevant project earlier), re-arrange the blocks of text accordingly. Ensure that moving sections doesn't break LaTeX syntax (each project entry should remain within its environment or list).
2. **Insert JD Keywords and Skills** – Enhance the resume text with important keywords from the job description:
    - Identify sections of the resume where keywords naturally fit. Common places are the **Skills section** (a list of technical skills, tools, etc.), the **Summary/Objective** (if the template has one), or within project descriptions.
    - Take the list of keywords the AI provided (or extracted by parsing the JD) and for each, check if it already exists in the resume text. If not, add it in a contextual way:
        - *Skills Section*: If the resume has a bullet or comma-separated list of skills, append any new ones here. For example, if "Kubernetes" and "CI/CD" were identified as missing keywords, add them to the skills list.

- *Project/Experience Bullets*: For keywords that relate strongly to an existing project, insert them into that project's description. E.g., if the JD emphasizes cloud deployment and the project description said "Deployed web services," it could be modified to "Deployed web services on AWS, using Kubernetes for container orchestration" to include the keyword.

- Ensure the additions feel natural and maintain the resume's professional tone. The goal is to **optimize for ATS** by including relevant terms from the JD linkedin.com without sacrificing readability.

3. **Incorporate AI-Suggested Edits** – Apply any other content changes recommended by the AI:

   - The AI might suggest phrasing improvements or emphasis on certain achievements. For instance, it might say "emphasize your leadership in Project X" or provide a reworded bullet point. Edit the LaTeX text accordingly (replace or augment existing lines) to reflect these suggestions.

   - If the AI output provided ready-to-use text snippets (and they are accurate), integrate those into the resume. Example: AI suggests changing "Implemented feature Y" to "Led the implementation of feature Y, resulting in 20% performance improvement" – update the line in the `.tex` file.

   - Maintain consistency in formatting (e.g. if all bullet points start with past-tense verbs, ensure new content follows the same style).

4. **Generate a New Project Description (if needed)** – If the AI recommended adding a completely new project or section to better match the JD:

   - Create a new project entry in the LaTeX file following the template's structure. For example, if the template uses `\headedsubsection{Project Name}{Duration}{Description}`, fill in these fields for the new project. Use a placeholder name or one inferred from the context (the AI might have given it a name or just described it).

- Use the AI's provided description for this project. If the AI only gave a concept (e.g. "show experience with machine learning model deployment"), and not the text, call the AI again with a prompt to *"Write a 2-3 line resume project description that demonstrates experience in X,"* where X is the skill or domain needed. Insert the resulting lines into the `.tex` file under a new project section.
- Mark this section as active (not commented), since it's meant to be included. If the base template had a dummy placeholder project commented out (like an "Additional Project" template), you could uncomment it and replace its content with this new info.

5. **Review LaTeX Syntax** – After all text modifications:
   - Check that the LaTeX file is still syntactically correct. Ensure that environments opened are closed, braces/brackets match up, and special characters (like % or _ in the content) are properly escaped if they are now in active text.
   - If the tool made structural changes (like moving sections or adding a new one), a quick sanity check can prevent compilation errors. This can be done by programmatically searching for common LaTeX section markers to ensure order (e.g., that every `\begin{itemize}` has a corresponding `\end{itemize}`, etc.).
   - Optionally, run a dry-run compile or a LaTeX linter in the background to catch any issues early (this can be integrated if LaTeX is available, or skip to Phase 6 and catch errors during actual PDF generation).

Phase 5: Change Diffing and User Review

1. **Compute Differences** – Before finalizing, present the user with a **diff** of the resume changes:
   - Use Python's `difflib` (or similar) to compare the original template and the modified resume text. This will yield a line-by-line difference, typically with `-` indicating removals and `+` indicating additions stackoverflow.com .

- Format the diff output in a clear way. For command-line display, you might color-code removed lines red and added lines green (using a library like `colorama` for terminal coloring). If color isn't available, the `+/-` prefixes and context lines should still be clear.
- Show a few lines of context around changes so the user sees the surrounding section for each edit.

2. **Display AI Rationale (Optional)** – If the AI provided rationale for changes (some structured outputs might include a reasoning or summary), display that alongside the diff. For example: "Included Project Alpha (+3 lines) because the JD emphasized cloud experience; Commented out Project Beta (-4 lines) as it's less relevant." This can help the user understand *why* changes were made.
3. **User Review Prompt** – Ask the user to review the diff and confirm the changes:
   - Provide a simple prompt like: *"Review the above changes. Enter (A)ccept to apply these modifications, (E)dit to manually edit the resume before proceeding, or (C)ancel to abort."*
   - If **Accepted**, proceed to Phase 6 (saving and compiling the resume).
   - If the user chooses to **Edit**, open the modified `.tex` in an editor (perhaps by launching the default text editor or allowing the user to specify one). This lets the user fine-tune wording or make any corrections by hand. After they save and close, you could re-run the diff for any new changes, then prompt accept again.
   - If **Canceled**, exit the program (or return to Phase 2/3 allowing them to retry with different settings).
4. **Incorporate User Edits** – If the user made manual edits, merge those into the in-memory representation of the resume (or simply re-read the `.tex` after they save). This ensures the final file reflects their adjustments.
5. **Ensure Satisfaction** – The goal is that the user is confident the new resume aligns with the job. This review step acts as a safeguard. Even when automated, human oversight is important to catch nuances or avoid any AI-introduced errors. As one guide suggests, after tailoring a resume with AI it's crucial to double-check the output meets your preferences before using it linkedin.com. Only once the diff looks good do we move on to save and generate the final files.

Phase 6: Finalizing Output (Saving LaTeX and PDF Generation)

1. **Save Updated LaTeX File** – Once changes are approved, write the modified resume content to the `.tex` file in the target directory:
    - Use the `<Company_Name>/<Role_Name>/resume.tex` path prepared in Phase 1. If a file already existed there and we're overwriting it, consider making a backup (though git history will also preserve it in the next phase).
    - Ensure the file write is successful (handle exceptions in case of permission issues or invalid path).
2. **Compile the LaTeX to PDF** – Invoke a LaTeX engine to produce the PDF resume:
    - Use a system call or library to run `pdflatex` (or `xelatex` as needed for the template) on the saved `.tex` file. For example, use Python's `subprocess.run(["pdflatex", "resume.tex"])` with the working directory set to the target folder.
    - If LaTeX compilation returns errors (non-zero exit code), capture the log/output. Print an error message for the user and halt (or return to editing if feasible). Common errors might be from bad syntax if the template was altered in an incompatible way.
    - On success, verify that `resume.pdf` is generated in the folder. Possibly run the compilation twice if the template uses references or complex formatting (though most resumes won't require multiple passes).
3. **Output the Result** – Inform the user of the saved files:
    - Print a confirmation like: *"Success! Customized resume saved to `Company/Role/resume.tex` and compiled as `resume.pdf`."*
    - Optionally, open the PDF automatically (if on a system with GUI, e.g. on macOS use the `open` command, on Windows use `os.startfile`, etc.) so the user can see the final formatted resume.
    - Because the PDF is the final product to send to employers, ensure it's up to date with the `.tex` (the compile step should have created it, but if for some reason it failed and was using an old PDF, alert the user to the failure).

4. **Maintain Folder Structure** – Ensure that all output files are neatly stored:
    - The `.tex` and `.pdf` for this company/role are together, which makes it easy to refer back to them. If additional assets were involved (images, etc., though unlikely for a resume), make sure they are in place so the PDF is self-contained.
    - Do not delete the base template – it remains the master copy for other applications. The user may have one base template used for all roles, which remains separate in its original location. The personalized copy now lives in the target folder.

Phase 7: Version Control Integration (Git Tracking)

1. **Initialize Git Repository (if needed)** – Since LaTeX files are plain text and work well with version control dev.to , integrate Git to track the evolution of resumes:
    - If the resume folder (or a higher-level directory encompassing multiple resumes) isn't already a git repository, initialize one. For example, call `git init` in the root folder that contains all the company subfolders.
    - If a repo already exists (perhaps the user's whole resume project is under git), skip init and proceed to adding files.
2. **Stage Changes** – Add the newly created/modified files to the git index:
    - Use GitPython library or system calls to stage `resume.tex` and `resume.pdf` for commit. For instance, with GitPython:
      `repo.index.add([resume_tex_path, resume_pdf_path])` gitpython.readthedocs.io .
    - If this is the first commit (no prior history), you may also stage the base template and any supporting files so the whole project is tracked.
    - Ensure that any intermediate or log files (LaTeX aux files, etc.) are ignored. Create a `.gitignore` if not present to exclude common LaTeX build files (`*.aux` , `*.log` , `*.out` , etc.) and maybe PDFs if the user doesn't want to version control binaries. (However, storing the PDF can be useful for quick retrieval, even though diffs will mainly focus on the .tex changes.)
3. **Commit Changes** – Create a git commit capturing the state of the tailored resume:

- Formulate a commit message that identifies the company and role, for example: `"feat: Tailored resume for [Role] at [Company]"` along with the date or job ID if applicable.
- Commit via GitPython (`repo.index.commit(message)`) or subprocess (`git commit -m "..."`). This snapshot includes the modifications made by the AI and any user edits.
- Each commit serves as a historical record. Later, the user can use `git log` or `git diff` to review how their resume tailoring has evolved over time for different applications.

4. **Tag or Branch (Optional)** – For organizational purposes, the tool could tag commits with the company name or role, or create a branch per company. This is optional but can help if the user is applying to many jobs:
   - For example, after committing, run `git tag TechCorp-SoftwareEngineer` on the commit. This makes it easy to jump back to that version.
   - Alternatively, use a branching strategy: main branch holds the base template, and each resume version lives in a branch named like `TechCorp-SoftwareEngineer` . This might be overkill for local use, so tagging or just commit messages is usually sufficient.
5. **Push to Remote (Optional)** – If the repository is linked to a remote (like GitHub or GitLab) and the user has set it up:
   - The tool can attempt to push the new commit to the remote (`git push origin main` , for example). Make sure to handle authentication (could use stored credentials or prompt the user).
   - Pushing is not strictly necessary for local functionality, but it provides off-site backup and the ability to share the resume changes if needed.
6. **Verify Git History** – After committing, for safety, you might print out a short log or diff:
   - e.g., run `git show --stat HEAD` to show that the commit was created and list the files changed, or `git log -1` to show the latest commit message.

- This gives the user confirmation that version control captured the update. LaTeX's compatibility with git ensures that every change (even a one-word tweak) can be tracked and reviewed later dev.to .

7. **Leverage Git for Learning** – (Optional) In future runs, the tool can programmatically query the git history to see past commit messages or diffs for similar roles:
    - For example, search commit messages for the same Company or Role to quickly find what was done previously (as an alternative to scanning the filesystem in Phase 3).
    - This reinforces the "learning from history" aspect by using version control data in addition to the saved files.

Expanding to a Basic Web Application

To make the resume personalization tool accessible via a browser, we can create a simple web app around the Python logic. Below are steps to extend the CLI tool into a web application, using Python for the backend and minimal JavaScript for interactivity:

1. **Choose a Web Framework** – Use a lightweight Python web framework such as **Flask** or **FastAPI** for the backend. This will handle HTTP requests and interface with the existing Python logic (which should be refactored into functions/modules that both the CLI and web can call).
2. **Design the Input Form (Frontend)** – Create an HTML form to collect the necessary inputs:
    - Fields for **Role Name** and **Company Name** (text inputs).
    - A textarea or file upload for the **Job Description**. For simplicity, a large textarea where the user can paste the JD may suffice. Alternatively, allow uploading a `.txt` or `.pdf` file (if file upload is desired, set the form's `enctype="multipart/form-data"` and handle file parsing on the server flask.palletsprojects.com).
    - A dropdown or radio buttons to select the **AI agent** (e.g. "ChatGPT" or "Gemini").
    - A submit button to send these inputs to the server for processing.

- Use basic HTML/CSS for layout; keep it simple and avoid heavy frameworks to align with the “minimal JavaScript” goal.

3. **Backend Route for Processing** – Define a Flask route (e.g. `/personalize`) to handle the form submission (POST request):
    - On form submission, the server retrieves the role, company, agent choice, and JD text from the request.
    - It then calls the internal functions that implement Phase 1–5 of the CLI logic: load template, call AI, apply changes, etc. (All the core steps can be reused; ensure any CLI-specific prompts are adapted or removed for non-interactive use).
    - Instead of printing a diff to console, prepare the diff as data to return to the user. For example, generate the unified diff string (from Phase 5) and store it.
    - The server should not automatically save and compile the resume yet; we’ll first show the diff to the user for approval in the web UI (mimicking the CLI review step).
    - Possibly maintain a session or temporary store (server-side, or via a signed cookie) to hold the modified `.tex` content or file until the user approves, since HTTP is stateless. This could simply be saving the modified `.tex` to a temp file or in a database keyed by session.
4. **Display the Diff on the Webpage** – After processing, the Flask route returns an HTML page showing the proposed changes:
    - Present the diff in a readable format. For minimal complexity, you can output the diff as `<pre>` text with `+` and `-` lines colored via simple CSS (green for additions, red for deletions).
    - Optionally, use a small JavaScript snippet to enhance this diff display (for example, you could use a library-free approach to toggle visibility of unchanged lines, or simply rely on the raw diff if clarity is sufficient).
    - Along with the diff, show buttons for **“Approve”** and **“Cancel”** (and possibly **“Edit”** if allowing manual tweaks in the browser).

- If an “Edit” feature is desired with minimal JS: you could include a textarea containing the entire modified resume content for the user to manually edit. However, this might be overwhelming. A simpler approach is to trust the diff and allow either approval or cancellation at this stage.

5. **Approval Handling** – If the user clicks “Approve”, send another request to the server to finalize the changes:
   - This could be done via a form POST or an AJAX call triggered by the approve button. (Using a standard form button keeps JS minimal: e.g., the Approve button is inside a `<form action="/finalize" method="POST">` so that clicking it submits to a finalize route.)
   - The server, upon receiving approval, will perform Phase 6 and 7: save the `.tex` file, run LaTeX to get the PDF, and commit to git. These steps can reuse the same code as the CLI, just ensure paths are correct (the server might run in a certain working directory).
   - After finalizing, the server can respond by redirecting the user to a download page or triggering a file download.
6. **Provide the Result to the User** – Once the PDF is generated, present a link or button for the user to download it:
   - For example, the finalize route could return a page with: “Your resume is ready! [Download PDF] [Download .tex]”. The PDF (and .tex if offered) would be served as static files from the output folder. Ensure the web server has access to serve files from the resume directory (you might configure Flask’s static file routing or use `send_file` to send the PDF).
   - Include a message reminding the user that the version is saved (and perhaps mention version control commit if relevant).
   - After download, the user can use the PDF as needed. The .tex is mainly for their record or further manual editing if they want.
7. **Minimal JavaScript Enhancements** – Keep the client-side simple, but a few JavaScript improvements can enhance usability:

- **Loading Indicator**: Since the AI analysis and LaTeX compilation can take a bit of time (several seconds or more), provide feedback during processing. This could be a simple JavaScript that triggers on form submission to show a "Please wait, personalizing your resume…" message or spinner. Alternatively, use HTML meta refresh or AJAX polling. A very minimal approach is a static "processing" page after form submission that auto-refreshes after a few seconds or instructs the user to wait.
- **Diff Interaction**: Use JS to allow the user to toggle whether they want to see the full diff or just a summary. For instance, a "Hide unchanged lines" checkbox that, when clicked, hides all lines that are context (neither + nor -). This can be done by wrapping diff lines in `<span>` with classes for added/removed/unchanged and using JS+CSS to show/hide. This keeps it lightweight (no large diff library needed) but is a nice convenience.
- **Form Validation**: Add a bit of JS to validate that the user has entered the required fields (role, company, and JD) before allowing submission, to avoid needless round-trips.
- Other than the above, the app can remain mostly server-driven. By avoiding a complex single-page app framework, we use minimal JS just to improve UX while Python handles the core logic.

8. **Reuse Business Logic** – Refactor the CLI code into modular functions that the web app can call:
   - For example, have a function `personalize_resume(jd_text, role, company, agent)` that executes Phases 2–5 and returns the diff and modified content. The CLI can call this and then do interactive prompts, while the web route calls it and then returns a webpage.
   - This avoids duplicating code and ensures any improvements to the logic benefit both interfaces.
   - Also, ensure error handling in this function is robust for the web scenario (you don't want stack traces exposed to the user; catch exceptions like AI API errors or LaTeX failures and return a user-friendly error page).
9. **Testing the Web App** – Before deploying, test the workflow:

- Try uploading or pasting various JDs, selecting different agents, and verify the diff shows sensible changes.
- Test the LaTeX compilation on the server environment to ensure the PDF generation works (the server may need LaTeX installed).
- Check that the git commits are happening on the server side (if the server is running locally, it will use the same repo mechanism; if on a different machine, you might skip git or have a separate repo clone there).
- Ensure that multiple users or requests won't conflict. For a basic app, you might assume single-user usage. If making multi-user, isolate each user's files, perhaps by naming output files with a session ID or using a database.

10. **Deployment Consideration** – If you intend to deploy this web app (beyond local use):
    - Use a proper WSGI server or containerize the application (for example, using Gunicorn for Flask).
    - Secure the application, especially since it might involve uploading resumes or JDs (which can be sensitive). At minimum, use HTTPS and consider an authentication mechanism if not meant to be public.
    - The minimal JS approach ensures the app is lightweight and easy to maintain, focusing on core functionality (AI + LaTeX) without a complex frontend. Users get the convenience of a web interface, while the backend automates the heavy lifting of resume customization, just as the CLI does.

Throughout this plan, each phase ensures that the resume is systematically tailored to the job description with AI assistance while keeping the user in control of the final content. By following these steps, we can implement a comprehensive tool that streamlines resume personalization and then broaden its accessibility via a simple web interface, combining the reliability of LaTeX formatting with the intelligence of modern AI models.

Sources: The approach builds on known practices of using LaTeX for consistent resume formatting and AI for content tailoring. LaTeX templates allow easy section toggling and work well with version control dev.to linkedin.com . AI agents like ChatGPT/Gemini can ingest job postings to identify key skills and suggest resume adjustments linkedin.com , including inserting important keywords to improve ATS rankings linkedin.com . In fact, one can feed a previous resume and the JD to ChatGPT to get a tailored resume and even a diff of changes linkedin.com , which is analogous to what this tool automates. Python's `difflib` provides a straightforward way to show textual differences stackoverflow.com , and Git integration (using libraries like GitPython) allows capturing each customized version for learning and rollback gitpython.readthedocs.io . These components together form a robust system for resume personalization and iterative improvement.

Citations

How My Laziness Led Me to LaTeX: Automating Resumes with AI
https://www.linkedin.com/pulse/how-my-laziness-led-me-latex-automating-resumes-ai-alisher-ataev-o7n2f

How My Laziness Led Me to LaTeX: Automating Resumes with AI
https://www.linkedin.com/pulse/how-my-laziness-led-me-latex-automating-resumes-ai-alisher-ataev-o7n2f

Integration of Google Gemini API as an Alternative to ChatGPT API · Issue #...
https://github.com/Significant-Gravitas/AutoGPT/issues/7902

Integration of Google Gemini API as an Alternative to ChatGPT API · Issue #...
https://github.com/Significant-Gravitas/AutoGPT/issues/7902

How to create your resume with chatGPT | Sami Sharaf | 166 comments
https://www.linkedin.com/posts/thesamisharaf_how-to-create-your-resume-with-chatgpt-activity-7221102990911303680-c4jJ

How My Laziness Led Me to LaTeX: Automating Resumes with AI
https://www.linkedin.com/pulse/how-my-laziness-led-me-latex-automating-resumes-ai-alisher-ataev-o7n2f

python - How to output difference between two text files? - Stack Overflow
https://stackoverflow.com/questions/25353284/how-to-output-difference-between-two-text-files

How to create your resume with chatGPT | Sami Sharaf | 166 comments
https://www.linkedin.com/posts/thesamisharaf_how-to-create-your-resume-with-chatgpt-activity-7221102990911303680-c4jJ

Crafting a Professional Resume with LaTeX and ChatGPT: A Step-by-Step Gu…
https://dev.to/aayush518/crafting-a-professional-resume-with-latex-and-chatgpt-a-step-by-step-guide-2f6h

GitPython Tutorial — GitPython 3.1.44 documentation
https://gitpython.readthedocs.io/en/stable/tutorial.html

Uploading Files — Flask Documentation (3.1.x)
https://flask.palletsprojects.com/en/stable/patterns/fileuploads/

How to create your resume with chatGPT | Sami Sharaf | 166 comments
https://www.linkedin.com/posts/thesamisharaf_how-to-create-your-resume-with-chatgpt-activity-7221102990911303680-c4jJ

All Sources

linkedin github stackoverflow dev gitpytho...adthedocs

flask.pa...sprojects