Auto-Filler AI

Project GithubLink:

Project Video Link:

Project Installation Link:

Introduction

The AI Form Filler project automates the process of filling out online job application forms and generating personalized cover letters for specific job descriptions. It uses user-provided documents—such as resumes and profile details (name, email, LinkedIn URL, GitHub URL, etc.)—to generate direct answers for straightforward fields and combines these details with job descriptions to craft role-specific answers. All information is dynamically fetched and processed using a Large Language Model (LLM), ensuring that responses to open-ended or motivational questions are relevant and tailored. Additionally, by inputting a job URL, the system extracts the job description and generates a custom cover letter that aligns the user's skills and experience with the job's requirements. Nothing is hardcoded; all answers are generated intelligently by the LLM.

Problem Statement

Context:
Filling out job application forms can be a tedious and time-consuming process for job seekers. While traditional browser-based auto-fill tools (like Chrome's auto-fill) assist with basic fields such as name, email, and address, they are incapable of answering more complex, role-specific, and motivational questions, such as "Why do you want this job?" or "What are your strengths?" These types of questions require thoughtful, personalized responses, often making the job application process repetitive and inefficient for candidates applying to multiple positions.

Problem:
The lack of advanced tools to automate and personalize the job application process presents a significant challenge. Current auto-fill systems do not address the need for intelligent, role-specific answers. This project leverages the capabilities of Large Language Models (LLMs) to not only auto-fill basic details but also generate contextual, personalized responses to job-specific questions, reducing the effort and time required for job applications.

Project Objectives

- **Objective 1**: Develop a browser extension that fills both basic personal details (e.g., name, email) and complex, job-specific fields in application forms.
- **Objective 2**: Integrate LLMs to craft tailored responses for questions such as "What makes you a good fit for this job?" or "Why do you want to work here?"
- **Objective 3**: Provide users with an intuitive interface to review, modify, or accept LLM-generated suggestions for accuracy and personalization.

Expected Outcome:
A functional tool that simplifies the job application process by efficiently filling out repetitive and job-specific questions. The system aims to enhance the overall user experience while maintaining personalization and relevance.

Methodology

To achieve this, the system leverages several key components:

1. **Large Language Models (LLMs)**:
   The project integrates two LLMs:
   - **GPT 4o mini**: This is a variant of an advanced language model that has proven to be efficient and responsive in the project. It provides high-quality, context-aware responses and serves as the primary model for generating answers. It successfully handled both direct and indirect responses, including open-ended and motivational questions.
   - **Gemini**: This Google Generative AI model was also integrated into the project. However, it did not perform as effectively as GPT-4o mini. Its outputs were less aligned with the project's specific requirements. While it could answer direct questions, it struggled to infer context or select the correct options for radio buttons or dropdown menus. As a result, GPT-4o mini became the primary choice for this task.

   The project uses public LLMs because the data involved is not personal. It primarily consists of publicly available resume information, which users often share on LinkedIn. Due to this, a personal LLM was not necessary. Although Ollama was also tested, it failed to generate accurate answers and couldn't output responses in the required JSON format. As a result, public LLMs were preferred, with a private secret key used for security.

2. **LangChain and Retrieval Augmented Generation (RAG)**:
   The project uses **LangChain** and implements **Retrieval-Augmented Generation (RAG)**.

   LangChain is a framework that helps build applications using Large Language Models (LLMs). It manages tasks like organizing prompts, linking different actions together, and integrating steps to retrieve information.

   RAG is a technique where the system first searches for relevant information (like details from the user's resume or a job description) and then feeds it into the LLM. This ensures that the answers generated by the LLM are based on real, available data rather than just the model's general knowledge.

   In simpler terms, RAG helps the LLM give more accurate and personalized responses. For example, if a form asks for a user's current city, the system uses RAG to find that information from the user's resume and provides the correct answer. This makes the responses more precise and true to the user's data.

3. **BeautifulSoup for Parsing HTML**:
   The system uses BeautifulSoup, a Python library for parsing HTML documents, to read the structure of the job application form pages. It identifies each input field, dropdown, or file upload element by analyzing the HTML. BeautifulSoup makes it easier to find the right elements and their labels, so the LLM knows exactly what to fill in.

4. **Selenium for Parsing Dynamic Content:**
   The system leverages **Selenium**, a browser automation tool, to parse job description URLs and extract content dynamically. This is especially useful for pages where job descriptions are rendered using JavaScript or where data is not immediately available in the HTML source. Selenium interacts with the

webpage as a user would, ensuring that all elements (such as dynamically loaded job descriptions) are captured for processing.

Detailed Workflow

1. **Loading and Processing the User's Information**:
   The user's resume and other related documents are loaded using PyPDFDirectoryLoader from LangChain to handle PDFs. The text is then split into smaller chunks using RecursiveCharacterTextSplitter. These chunks are then converted into vector embeddings using HuggingFaceEmbeddings.

   The embeddings are stored in a FAISS vector store, which allows fast similarity searches. When a question arises, like "What is the user's GitHub URL?", the system can quickly retrieve the most relevant chunk of text from the resume and feed it into the LLM.

2. **Retrieval Augmented Generation (RAG) in Action**:
   When filling forms or generating cover letters, the system does the following:
   - Queries the vector store to find relevant information about the user (e.g., their location, degree, or experience).
   - Passes that retrieved text as context to the LLM (GPT 4o mini) along with the prompt.

   This ensures the responses are not guesses but informed by the user's actual data. For example, if the form asks if the user is authorized to work in the United States, the system finds that detail in the resume document and instructs the LLM to include it.

3. **Filling Out the Form**:
   After the system knows what the form's fields are (extracted via BeautifulSoup), it uses a carefully crafted prompt to tell the LLM how to fill each field. It can handle:
   - Simple text fields: directly fill with the user's info.
   - Radio buttons and dropdowns: select the correct option from the provided list.
   - File attachments: attach the user's resume by simulating a user file upload.
   - Open-ended questions: produce thoughtful answers based on the user's values, motivations, and background.

   The final answers are returned as a JSON object. A client-side script (e.g., in popup.js and content.js) then uses this JSON to automatically input the data into the web form.

4. **Generating a Cover Letter**:
   For cover letter generation, the user provides a job URL containing the job description. The system fetches the job posting's HTML content, extracts the text (which includes role requirements and responsibilities), and feeds it into the LLM along with the user's resume data.

   The LLM is prompted to write a professional cover letter that aligns the user's experience with the job. It ends with "Sincerely," followed by the user's real name—no placeholders. If the user's name isn't found in the document, it avoids writing a name altogether.

The Prompts Used

Below are the prompts exactly as written in the code. These prompts guide the LLM in how to respond.

Prompt for Form Filling (filling_form_single_request):

```
prompt = (
        f"You are an AI assistant helping to fill out a job application form using the provided documents. I am sharing my resume, "
            f"which contains my name, email, phone number, LinkedIn URL, GitHub URL, and portfolio URL (https://varun-sahni.com/), but these details might not have explicit labels. "
        f"Extract this information directly from the text in the resume and map it appropriately to the form fields. "
        f"Here is the form structure and its valid options (if applicable):\n\n"
        f"{json.dumps(json_request, indent=2)}\n\n"
       f"For the field with the key 'No location found. Try entering a different locationLoading', if a location is present in the document, write the location from the document. "
            f"For each question, generate an appropriate response based on the user's resume, documents, and job application context. "
       f"Ensure that fields such as LinkedIn URL, GitHub URL, Twitter URL, and portfolio URL are filled using the provided information. "
            f"If specific data for these fields is missing in the documents, infer or generate placeholder URLs (e.g., 'https://www.linkedin.com/in/username', "
        f"'https://github.com/username', 'https://twitter.com/username', 'https://yourportfolio.com') based on standard formats, ensuring relevance and professionalism. "
        f"If a question asks about motivations, company-specific enthusiasm, or open-ended responses (e.g., 'What gets you excited about joining this team?'), "
        f"generate a thoughtful answer based on common professional aspirations and values. "
       f"For any questions not explicitly covered in the document, provide a response based on your knowledge about me and the context of the job application. "
        f"Ensure that all responses strictly conform to the options provided (if any). "
        f"Return the completed JSON object strictly in JSON format without any additional text, code blocks, or comments."
    )
```

Prompt for Cover Letter Generation (create_cover_letter):

```
prompt = (
```

```
        f"Using the following job description and the provided documents, write the body of a fully tailored, polished, and professional cover letter. "
        f"Ensure the output is a complete, ready-to-submit cover letter body that focuses on aligning my experience, skills, and education with the job's requirements and the company's mission and values. "
        f"End the letter with 'Sincerely,' followed by my full name, which you will extract directly from the provided document. "
        f"Do not include any placeholders or guess my name if it is missing. If the name is not found in the document, do not write closing salution."
        f"Do not mention the platform where the job was found, specific hiring manager names, or any placeholders."
        f"If specific job details are missing, craft a general professional cover letter body suitable for a technical software engineering role, emphasizing my key strengths and achievements. "
        f"Job Description:\n{job_description}\n\n"
        f"Cover Letter Body:"
    )
```

These prompts are designed to be very explicit. They tell the LLM what to do step-by-step, ensuring the final output is exactly what the application needs.

Measuring Tokens and Time

To understand the performance and cost, the project measures:

- **Number of tokens** in the prompt and in the LLM's response.
- **Time taken** by the LLM to produce the answer.

Tokens are a way LLMs count text usage. By measuring them, we know how large our requests and responses are. This helps with optimization and budgeting. The project uses the tiktoken library for token counting. It also prints the response time, so we know how quickly the LLM answers. This can be important if we integrate more complex functionalities or want to run this tool at scale.

Here's what is printed out in the console:

- Prompt Tokens: How many tokens the prompt consumed.
- Response Tokens: How many tokens the LLM response consumed.
- Total Tokens: The sum of both.
- Response Time: How many seconds it took for the LLM to return a complete response.

These details help in understanding performance and making future improvements.

Installation Guide

Follow these steps to set up the Auto Filler AI system and Chrome extension.

---

1. Extract the ZIP Folder

- Download the project ZIP file and extract it to your desired location on your computer.

2. Create Your Info Document

- Inside the extracted folder, you'll find a Word document named document.docx in the home directory. This document contains the template that was used for input. You can edit this document, enter your details, generate a PDF, and then place it into the info directory inside the main project directory.
- Once done, make sure to remove document.pdf from the info directory, as it is the previous document that contains my details. The LLM will use the information in your new document.pdf that contains your details to process and generate responses.

3. Set Up Virtual Environment

- Open a terminal or command prompt in the root directory of the extracted folder.
- Create a new Python virtual environment by running the following command:

  python -m venv venv

- Activate the virtual environment:
  - On Windows:

    venv\Scripts\activate

  - On macOS/Linux:

    source venv/bin/activate

4. Install Dependencies

- Once the virtual environment is activated, install the required Python dependencies:

  pip install -r requirements.txt

- If any additional dependencies are needed, install them manually using pip:

  pip install <package-name>

5. Run the Backend

- Now that your environment is set up, run the backend server by executing:

  python Auto_filler_AI.py

- This will start the backend process and keep it running to support the Chrome extension.

---

6. Install the Chrome Extension

- Open Google Chrome and go to the Extensions page by navigating to chrome://extensions/:
  - Chrome > Settings > Extensions

- Enable "Developer mode" in the top-right corner.
- Click "Load Unpacked" and select the chrome-extension folder located inside the root directory of the project.
- The extension will now be installed and available in your browser. You can pin the extension icon to the address bar for easier access.

---

7. Using the Extension

- **Filling a Cover Letter:**
    - Navigate to a job description webpage.
    - Copy the URL of the page containing the job description.
    - Open the extension and paste the URL into the provided field.
    - Click "Generate Cover Letter." The extension will generate a cover letter based on the job description.
- **Filling Out a Form:**
    - To fill out a form on a website, you can either upload your resume from your local computer or provide a Google Drive link to your resume.
    - Once the resume is attached, or the URL is entered, click on the "Fill Form" button.
    - The extension will autofill the form fields using the resume data.
    - After filling out the form, you can manually review the fields and make any necessary adjustments.
    - Once satisfied, click "Submit" to submit the form.

Troubleshooting

If you encounter any issues during the installation or usage, make sure the backend server (Auto_filler_AI.py) is running. Ensure that the virtual environment is activated and all required dependencies are installed. If the issue persists, refer to the installation video installation_video.mp4 in the project's main directory for a step-by-step guide. Additionally, you can consult the README.md file for further help.

Current Effectiveness and Scope

Right now, this tool is most effective on job application portals that use a form structure similar to **Lever**. Lever is a job application hosting platform used by many companies. The code is designed to handle simple HTML forms, and during testing, it successfully filled all forms on Lever. This effectiveness could be due to Lever's consistent form structure or the correct placement of tags, which follow a generic template.

However, for other portals with very different or more complex JavaScript-driven interfaces, the tool may not work as seamlessly. For example, some job portals dynamically load fields or have unusual HTML structures, which makes

it harder for BeautifulSoup to parse them reliably. For broader compatibility, more advanced web scraping techniques or browser automation tools might be needed.

Using GPT-4o Mini and Gemini

As noted, GPT 4o mini performed well, providing coherent and context-aware answers. Gemini integration was tested, but the quality of responses was not as good. While Gemini is a promising model, in this specific project GPT 4o mini was the better choice.

This shows that not all LLMs perform the same way in all tasks. Sometimes, a model might not handle the given domain or instructions well, and it's important to test and choose the model that best fits the needs.

User Interface and Flow

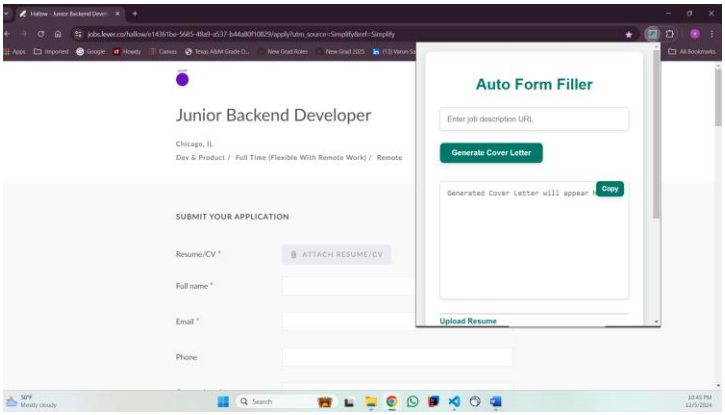
The project also includes a user interface (UI) component, using a browser extension setup (with popup.js, content.js, and background.js). The user can select a resume file or URL, then click a button to fill a form automatically. For cover letter generation, they can provide a job URL and get a custom cover letter in a text area. There is also a copy button to copy the cover letter.

Live Screenshots:

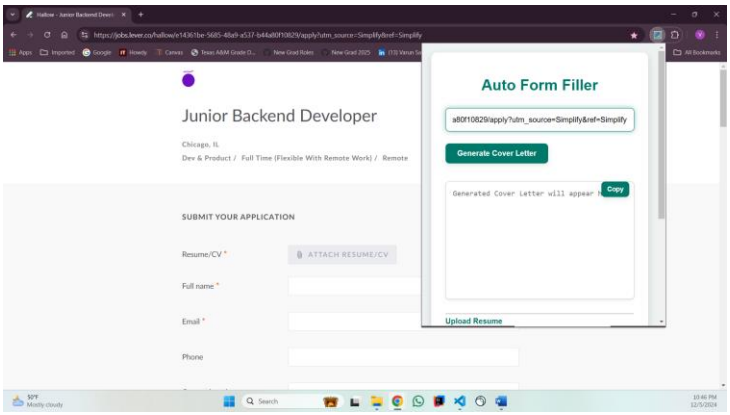
The following are the live screenshots demonstrating the functionality of the tool:

Cover Letter Generation:

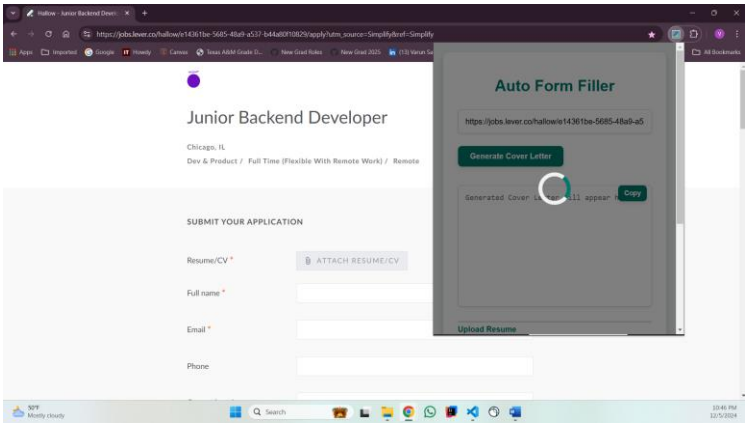
Screenshot 1: Interface asking the user to enter a job description URL.



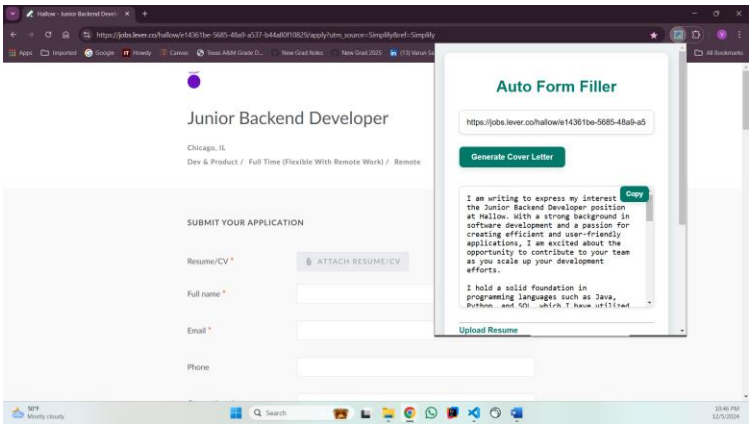
Screenshot 2: URL entered and the "Generate" button to be clicked.



Screenshot 3: Loader indicating that the cover letter is being generated.

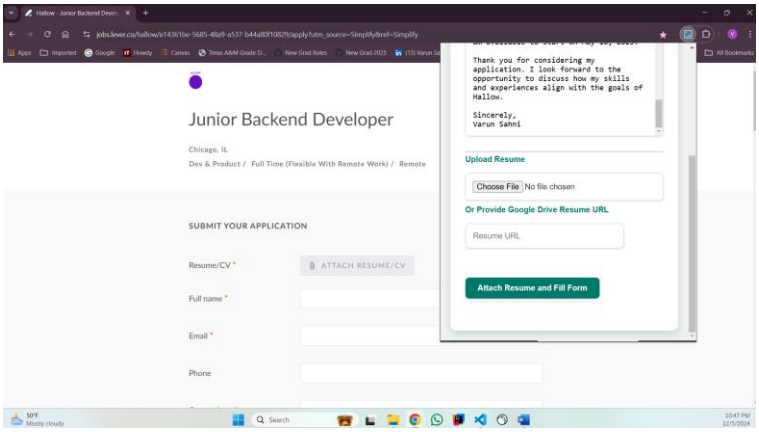


Screenshot 4: Final generated cover letter displayed in the extension.

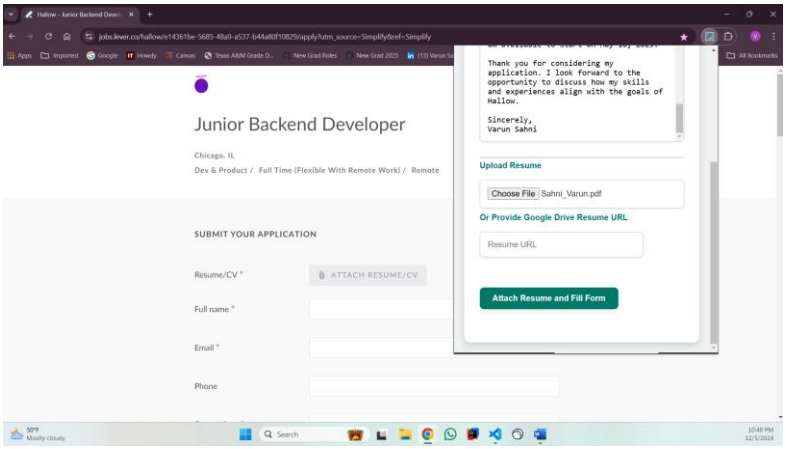


Form Filler:

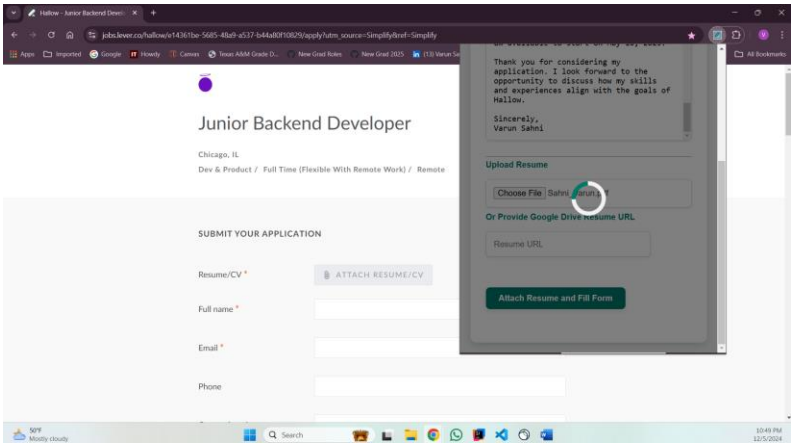
Screenshot 1: Initial screen asking the user to attach a resume (either by uploading or entering a URL).



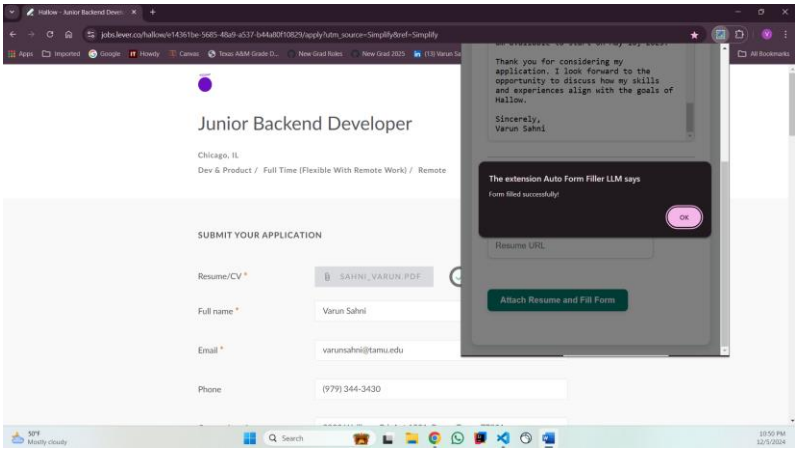
Screenshot 2: Resume attached, and the "Fill Form" button to be clicked.

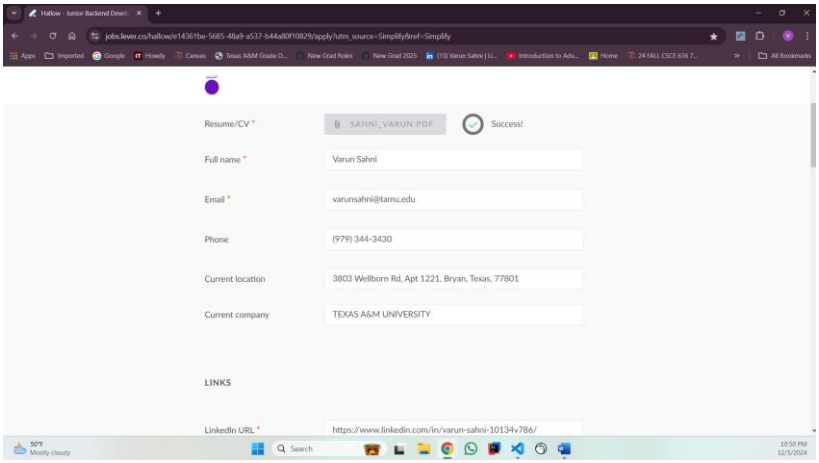


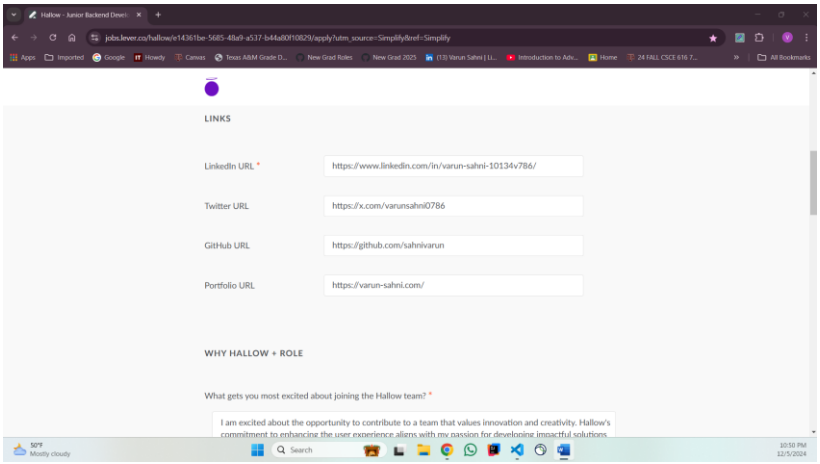
Screenshot 3: Attach Resume and Fill Form button clicked, Loader starts loading.

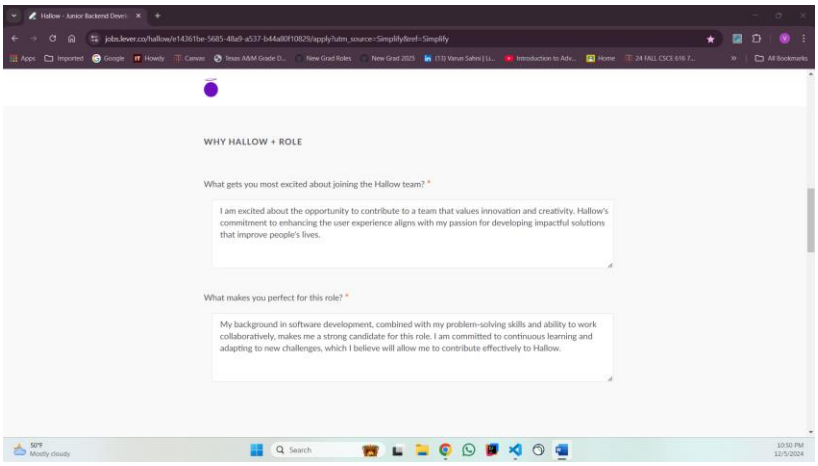


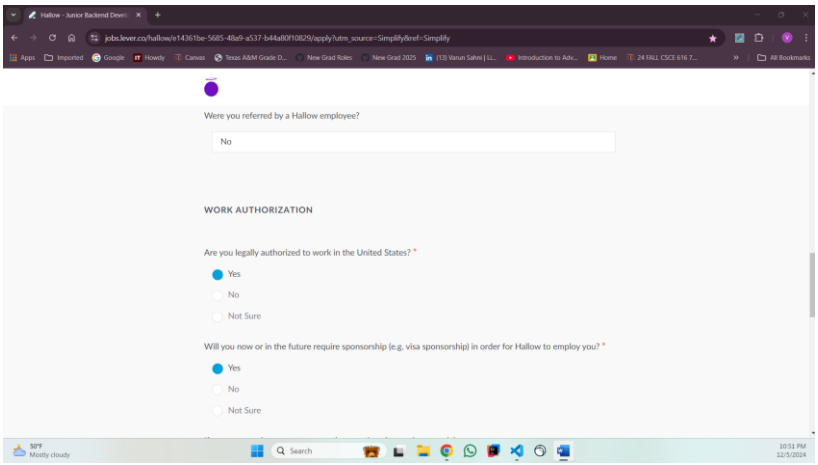
Screenshot 4: A filled web form, showcasing how the tool populates the fields with the user's information.

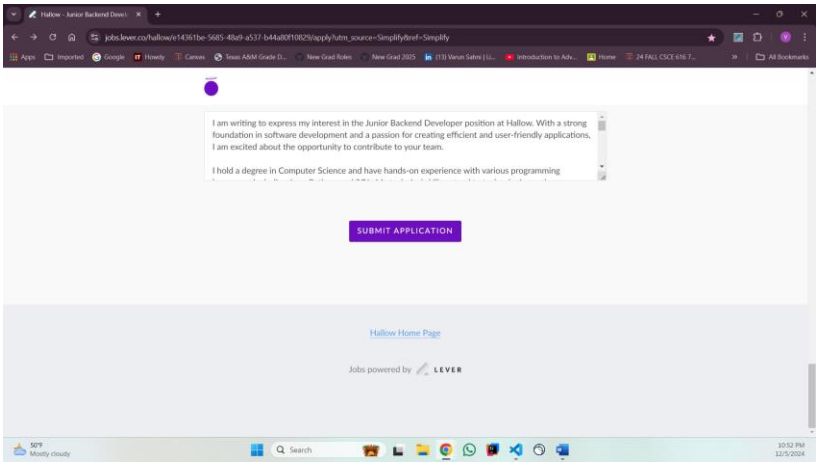


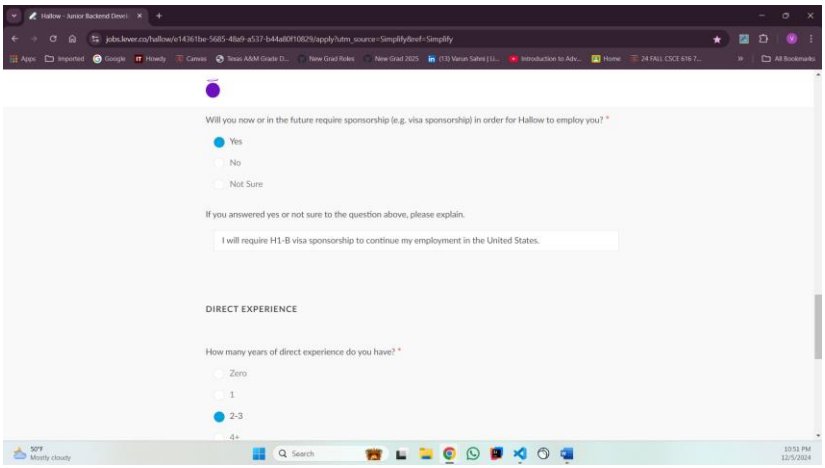


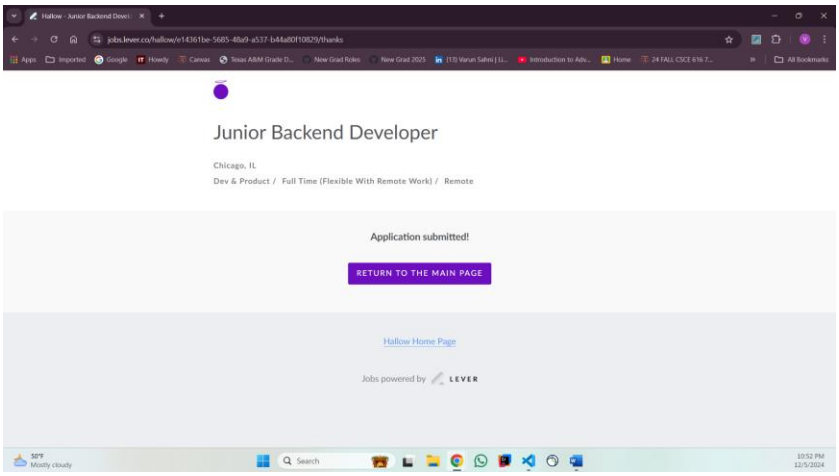



WHY HALLOW + ROLE
What gets you most excited about joining the Hallow team? *
I am excited about the opportunity to contribute to a team that values innovation and creativity. Hallow's commitment to enhancing the user experience aligns with my passage for developing impactful solutions that improve people's lives.
What makes you perfect for this role? *
My background in software development, combined with my problem-solving skills and ability to work collaboratively, makes me a strong candidate for this role. I am committed to continuous learning and adapting to new challenges, which I believe will allow me to contribute effectively to Hallow.


Were you referred by a Hallow employee?
No
WORK AUTHORIZATION
Are you legally authorized to work in the United States? *
Yes
No
Not Sure
Will you now or in the future require sponsorship (e.g. visa sponsorship) in order for Hallow to employ you? *
Yes
No
Not Sure


I am writing to express my interest in the Junior Backend Developer position at Hallow. With a strong foundation in software development and a passion for creating efficient and user-friendly applications, I am excited about the opportunity to contribute to your team.
I hold a degree in Computer Science and have hands-on experience with various programming
SUBMIT APPLICATION
Hallow Home Page
Jobs powered by LEVER





These screenshots illustrate the step-by-step functionality of the tool in real-time.

Sample Webforms Tested

Under the Results and Evaluations section, the tool was tested on various job application webforms hosted on Lever. Below are some of the webforms used during testing:

1. Hallow
2. Genesis Therapeutics
3. WHOOP
4. WHOOP (Position 2)

These forms contained a range of questions, from text-based inputs to dropdown menus and file uploads. The tool performed well in parsing and filling fields for these forms.

Evaluations and Results

The project underwent thorough testing to evaluate its performance across various scenarios. Here are the detailed findings:

1. Performance on Lever Platform

We tested the tool on 10 web pages hosted on Lever, containing basic job application questions. The results were largely positive:

- **Success Rate:** The tool successfully filled most fields on all 10 forms. However, minor issues, such as dropdown misalignment or missing field options, were observed in 1-2 questions per form.
- **Issues Observed:**
    - For 2 out of 10 forms, questions like "Race," "Gender," and "Disability" were not filled initially. However, on reattempting, these fields were filled correctly.
    - Upon analyzing these 2 failure cases, we found that the LLM occasionally added an additional character (e.g., an unnecessary comma) or selected a similar but incorrect response.
    - Example: For "Veteran Status," the LLM provided the response *"No, I am not a Veteran"*; however, the dropdown field only had the option *"No, I am not a protected Veteran."* As a result, the field remained unfilled.

2. Testing Gemini LLM

The Gemini LLM was also tested, but its performance was subpar:

- It consistently failed to generate the exact format required for dropdown and radio button questions.
- Issues observed:
    - Improper JSON formatting for fields with numerous options.
    - Difficulty in choosing the exact option among multiple choices.
- While Gemini could answer textual questions (e.g., open-ended questions), it failed entirely on dropdown and radio button questions.
- Failure Rate: All 10 Lever forms failed when using Gemini for these specific field types (radio and drop-downs).

3. Performance on Non-Lever Platforms

The models were also tested on other job application portals.

- **Key Findings:**
    - LLMs (both GPT 4o mini and Gemini) were prepared to parse and answer questions, but the BeautifulSoup parser struggled to parse forms with complex HTML structures.
    - Reasons for failure included:
        - Nested div elements.
        - Complex use of labels or reliance on non-standard identifiers (e.g., id attributes) that the LLM cannot infer without proper field mapping.

4. Specific Complex Cases

The tool was tested on a form with 200+ educational school choices.

- Example: Valkyrie Trading Form and Palantir Form.
    - In both cases, the app could not select the correct school from the dropdown due to the extensive number of options.
    - GPT 4o mini failed to identify the exact match, and the application occasionally froze during processing.

5. Token and Time Analysis

We measured the average time and token usage for both form filling and cover letter generation by GPT 4o-mini:

- Form Filling:
    - Average Prompt Tokens: 750
    - Average Response Tokens: 650
    - Average Total Tokens: 1400
    - Average Response Time: 10 seconds
- Cover Letter Generation:
    - Average Prompt Tokens: 105,000
    - Average Response Tokens: 300
    - Average Total Tokens: 105,300
    - Average Response Time: 15 seconds

These evaluations highlight that while GPT 4o mini is highly effective in most scenarios, edge cases involving extensive options or non-standard HTML structures remain challenging for the current implementation. Future improvements in web scraping and field detection, coupled with enhanced model fine-tuning, could address these issues.

Conclusion

This project shows how tools like Large Language Models (LLMs), LangChain, RAG, and web-scraping libraries such as BeautifulSoup and Selenium can work together to tackle a real-world problem: making job applications easier and faster. By combining these technologies with user details and job descriptions, the system can fill forms and create customized cover letters efficiently.

Key Highlights:

- **Efficiency:** The tool saves time by automating the process of filling job application forms and generating cover letters.
- **Personalization:** Answers and cover letters are tailored using the user's details, ensuring they are relevant and professional.
- **Scalability:** While the tool is currently best suited for Lever-based forms, it can be improved to handle other platforms with updates to its parsing and integration methods.

In conclusion, the success of this system depends a lot on the LLM's ability to provide accurate responses. Sometimes, the LLM might not give perfectly structured answers or choose the right options for complex fields. But this project wasn't about hardcoding solutions—it was about exploring the power of programming LLMs. By using methods like RAG and LangChain, we aimed to solve a real-world problem in a modern and meaningful way.

With more improvements, this tool could become a trusted companion for job seekers, making their journey a little easier and less stressful.

Challenges Faced and Risks

During the development of the AI Form Filler, a few key challenges were encountered:

1. **Web Scraping**
    - Tools like BeautifulSoup and Selenium were used to parse web pages.
    - Challenge: Complex job portals with dynamic JavaScript and nested elements were difficult to scrape reliably.
2. **Using LLMs**
    - Gemini, a Google AI model, failed to produce accurate results, especially for dropdowns and radio buttons.
    - Solution: Switched to GPT 4o mini, which was more reliable but required a purchase.
3. **Mapping JSON Responses to Form Fields**
    - The LLM sometimes gave responses that were close but not exact matches for dropdown or radio button options, causing fields to remain unfilled.
4. **Attaching Resumes**
    - Resumes needed to be converted into a format suitable for attaching to forms, adding extra steps.
5. **Risk: Data Privacy**

- Resume data is sent to a public LLM for processing, which could pose privacy concerns. However, this risk is mitigated by the fact that resumes are often publicly available on platforms like LinkedIn. Additionally, the project ensures secure API usage with private keys, minimizing data exposure risks.

Despite these challenges and risks, the system functions effectively and serves as a strong foundation for further improvement.

Future Improvements

There is a lot of potential to expand this project:

1. **Support More Job Portals**:
   Right now, the system works best on Lever-based application forms. We can add logic to handle other portals by improving web scraping and field detection methods to account for more diverse HTML structures or dynamic JavaScript-driven interfaces. However, the LLM itself is capable of answering most of the questions accurately, as tested. The limitations are primarily due to challenges in parsing the web form and correctly extracting and mapping the required fields in the request. Addressing these challenges would significantly expand the tool's compatibility and usability.

2. **Better Error Handling**:
   Improving error messages and fallback strategies if the LLM returns invalid JSON.

3. **More Advanced Personalization**:
   The cover letter generation could be enhanced to match specific industries, add introductions, or adapt tone and style based on user preferences.

4. **GUI Enhancements**:
   Adding a rich interface, preview windows, or step-by-step guidance would make it more user-friendly.

5. **Real-Time Feedback**:
   As the user changes their resume or picks different job URLs, the system could dynamically update suggestions or highlight which fields it can fill.
