Automated Internship Posting for Job Sites with Selenium and GenAI

***Abstract*—In India, millions of capable students graduate every year, and internships serve as critical stepping stones toward employment and skill development. However, many deserving candidates do not secure internships due to lack of visibility in a crowded market. To address this issue, an AI-driven automation system has been designed in this work to increase the volume and relevance of student applications. The solution begins by analyzing the user's resume using large language models (LLM) that extracts skills and preferences. A Selenium-based automation script then logs into job portal using credentials stored securely in a .ENV file. After a manual click to pass bot-detection, the system scrapes relevant internships and auto-fills applications using context-aware data generated by Gemini 1.5 Flash. Before finalizing, each application is paused for user review, ensuring accuracy and minimizing errors. This hybrid model allows students to apply to hundreds of tailored roles within minutes of time, significantly boosting their chances of being shortlisted.**

***Index Terms*—AI-driven opportunities, Candidate visibility, Gemini 1.5 flash, Internship automation, Large language models (LLMs), Resume analysis, Selenium, and Skill extraction.**

I. Introduction

Large Language Models (LLMs) are a class of advanced artificial intelligence (AI) systems built on foundation models trained on massive datasets. They have transformed to understand and generate human language, enabling applications in education, business, research, and creative fields. Unlike traditional models that required task-specific training, LLMs can handle multiple tasks with minimal fine-tuning, making them more scalable and cost-effective. Their rise coincides with breakthroughs in deep learning, especially transformer-based architectures that power today's natural language processing systems.

Since OpenAI's GPT-1 in 2018, with 117 million parameters, LLMs have grown rapidly in size and capability. The further versions GPT-2 with 1.5 billion parameters, GPT-3 reached to 175 billion, and GPT-4 has launched in 2023 with trillions of parameters, offering major improvements in reasoning and factual accuracy. GPT-4 Turbo, launched later that year with features such as optimized performance and cost, though its size remains undisclosed. LLMs are now widely accessible through open AI tools. Google's Gemini models add to this ecosystem with their multi-modal capabilities such as processing text, images, audio, code, and video. Integrated into products like Gmail, Google Docs, and Pixel devices, Gemini supports tasks like content generation and smart search. As these systems evolve, they are reshaping human-machine interaction across everyday digital experiences.

Securing internships has become an increasingly challenging endeavour for university students, not due to a shortage of skills or qualifications but, primarily because of under-application and the overwhelming, fragmented nature of internship discovery platforms. Many students, despite being capable and eligible, miss out on opportunities simply because they do not apply widely or consistently enough. Based on this insight, this paper presents an intelligent, automated system that seeks to bridge this gap by combining the capabilities of LLMs with web automation tools to streamline the internship application process. The system begins by allowing the user to submit their resume via a prompt interface. A fine-tuned LLM then reads and understands the resume, extracting key information such as skills, interests, and educational background. Using this context, the system initiates a Selenium-driven automation pipeline that installs a WebDriver, opens the job posting website, and logs in using credentials securely stored in a .env file. The automation process is designed to mimic human interaction, entering search filters and preferences based on the resume analysis, while pausing for manual verification to ensure compliance with platform called, bot detection protocols. Once access to the platform is fully granted, the system dynamically scans internship listings in real-time. It intelligently filters out irrelevant listings and selects only those that align with the user's background and goals. At this stage, Gemini 1.5 Flash, a powerful open source LLM tool, is employed to auto-fill the application forms with contextually appropriate and personalized information. Each application is drafted to present the candidate positively and accurately, and before any final submission, the system halts for a manual review. This deliberate human-in-the-loop design ensures the reliability of applications and minimizes the impact of potential LLM hallucinations or inaccuracies. This hybrid architecture enables the user to apply to hundreds of tailored internships within a short time frame. It dramatically increasing the odds of receiving interview

calls and being shortlisted. The paper demonstrates how Ai, and automation can be practically applied to real-world challenges faced by students. The system empowers students to act on more opportunities validating the belief by reducing the time and effort required to find and apply for suitable internships.

This paper is structured as follows: Section II contains the literature review to highlight existing approaches and identify research gaps. The proposed methodology designed to address some of the gaps has been proposed in section III. Section IV discusses the results and observations. Finally, Section V concludes the study with references to the works that have supported our research.

II. Literature Review

Recent advancements in web scraping have led to the development of diverse tools and systems that vary in complexity, adaptability, and integration capabilities. [1] lays the groundwork with a comparative analysis of popular scraping tools such as BeautifulSoup, Selenium, and Scrapy, benchmarking them across different browsers and assessing their efficiency, scalability, and ease of integration. Complementing this, [2] delves into Python-based libraries like requests and Selenium, investigating various configurations including headless mode, user-agent rotation, and ethical scraping practices to optimize for dynamic web content. Building on practical implementations, [3] demonstrates an end-to-end Selenium-based pipeline that scrapes product listings and feeds them into a machine learning model for price prediction, emphasizing the seamless fusion of data collection and predictive analytics. A research-focused perspective is provided by [4], which integrates Selenium with BERT to automate academic paper retrieval and semantic analysis, showcasing the synergy between scraping and NLP in knowledge extraction. Addressing real-world challenges such as CAPTCHAs, IP bans, and DOM volatility, [5] proposes design patterns and algorithmic strategies to ensure resilient, ethical scraping pipelines suitable for long-term deployment. Moreover, the utility of scraped data is extended in behavioral and commercial contexts. [6] demonstrates how social media metrics scraped from the web can be transformed into visual narratives for behavioral analytics, while [7] presents a modular Amazon product scraper that captures key attributes and renders them through interactive dashboards for business insights. Scraping also plays a critical role in compliance and brand integrity, as illustrated by [8], whose system detects counterfeit listings across e-commerce sites by comparing scraped data against verified records. In the telecom sector, [9] offers a Selenium-powered solution that aggregates regional plans and data offerings, storing them in structured formats with visual comparisons. Finally, pushing the boundary of accessibility, [10] proposes a natural language user interface for scraping that combines BERT with voice or text queries, allowing non-technical users to initiate complex scraping workflows through intuitive prompts. Collectively, these studies showcase the evolving landscape of web scraping from foundational tools and resilient architectures to intelligent, user friendly interfaces highlighting its multifaceted role in automation, analytics, and real-time information retrieval.

III. Proposed Methodology

The Proposed Methodology involves combining Selenium based web automation with Large Language Models like Gemini-1.5-flash for powerful and intelligent decision making for automated internship applications.The process is modular and consists of several steps:

A. Resume Upload

The user begins the procedure by uploading their resume, which can be in either a .pdf or .docx format. This file serves as the primary source of information about the candidate. It typically contains sections such as education, work experience, skills, projects, and certifications.

B. Text Extraction Methods

For machine-readable documents, prompt-based parsing techniques are used. This involves reading the textual content directly from the document using libraries such as python-docx or pdfminer.

For image-based or scanned documents, OCR is applied using Tesseract. OCR identifies and extracts characters from images, converting them into machine-readable text.

- T_{prompt}: Text extracted using prompt parsing.
- T_{ocr}: Text extracted using OCR.

To measure the consistency between the two methods, we calculate the **Jaccard Similarity Index**:

$$Accuracy_{text} = \frac{\left|T_{prompt} \cap T_{ocr}\right|}{\left|T_{prompt} \cup T_{ocr}\right|} \tag{1}$$

This helps in validating the fidelity of OCR output and ensures the correct data is used in subsequent stages.

C. Credential Input and Secure Storage

1) User Credentials: To facilitate secure access to job application portals, user credentials comprising a `username` and `password` are stored in a `.env` file. These credentials are retrieved using the `python-dotenv` module to ensure security and environmental abstraction. The relevant code snippet is as follows:

```
from dotenv import load_dotenv
import os
```

```
load_dotenv()
USERNAME = os.getenv("USERNAME")
PASSWORD = os.getenv("PASSWORD")
```

2) Form Population: Selenium WebDriver is utilized to automate the login process by populating form fields dynamically. The credentials are injected into the login form using the `send_keys()` method, as shown below:

```
driver = webdriver.Chrome()
wait = WebDriverWait(driver, delay)
driver.get("job-posting-website.com")
email_field = wait.until(EC.presence_of
_element_located((By.XPATH,
"//input[@type='email']")))
email_field.send_keys(USERNAME)
password_field = wait.until(EC.
presence_of_element_located((By.XPATH,
"//input[@type='password']")))
password_field.send_keys(PASSWORD)
```

The login button is clicked manually to bypass potential CAPTCHA or two-factor authentication measures.

3) Security Considerations: Credentials are abstracted via environment variables to prevent hard-coding sensitive information. Selenium ensures form fields are dynamically identified and populated, reducing the risk of DOM structure changes. Manual intervention is included to account for multi-factor authentication steps.

D. Session Initialization and Login Process

Browser Setup

- A **headless browser** (Chrome or Firefox) is initialized for automated tasks.
- A **visible browser** is also prepared in parallel for manual intervention where CAPTCHA or multifactor authentication is required.

Time Logging

The time taken for login is calculated to evaluate system efficiency:

$$T_{\text{login}} = t_{\text{submit}} - t_{\text{start}} \tag{2}$$

Where t_{start} and t_{submit} are the timestamps when form population starts and when the login is successfully completed.

E. Scraping of Job Postings

DOM Analysis

Using Selenium, the bot scans the Document Object Model (DOM) to identify tags (e.g., `<div>`, `<section>`) that contain job listings.

Let:

- $J = \{j_1, j_2, \ldots, j_n\}$: Set of job postings scraped from the page. Each job j_i contains fields like:

$$j_i = \{\text{title}_i, \text{desc}_i, \text{skills}_i, \text{location}_i\}$$

These are further processed in later stages to evaluate their relevance to the user's resume.

F. Relevance Score Calculation

Resume and Job Representation

Both the resume and job descriptions are transformed into numerical vectors using NLP techniques:

- **TF-IDF Vectorization**: Emphasizes unique terms.
- **Transformer-based Embeddings (e.g., BERT)**: Captures contextual meaning.

Similarity Computation

Cosine Similarity:

$$\text{Sim}(T_r, T_j) = \frac{T_r \cdot T_j}{\|T_r\| \|T_j\|} \tag{3}$$

Where T_r is the resume vector and T_j is the job description vector.

Keyword Overlap Score:

$$\text{Overlap} = \frac{|K_r \cap K_j|}{|K_j|} \tag{4}$$

Where K_r and K_j are sets of keywords from the resume and job posting respectively.

Composite Relevance Score

$$R_i = \alpha \cdot \text{Sim}(T_r, T_j) + \beta \cdot \text{Overlap}(K_r, K_j) \tag{5}$$

Where $\alpha + \beta = 1$ and weights are tuned based on empirical results.

Filtering Based on Relevance

$$J_{\text{filtered}} = \{j_i \in J \mid R_i \geq \tau\} \tag{6}$$

his filters out low-relevance postings before proceeding to form filling.

G. Application Form Filling via LLM (Gemini-1.5-Flash)

Input to Gemini

Each filtered job posting j_i generates questions in an application form. These questions are fed to Gemini-1.5-Flash along with the resume context.

Response Generation

$$A_i = \text{LLM}_{\text{Gemini}}(Q_i \mid T_r) \tag{7}$$

The LLM is tuned to generate context-aware and always-positive answers suitable for internship/job forms.

Form Interaction

Each generated answer a_i is entered into the appropriate form field f_i using Selenium.

H. Manual Review and User Confirmation

Before final submission, the user is given an interface to:

- Review generated answers.
- Edit responses if necessary.
- Confirm the application.

Time taken in this step is:

$$T_{\text{review}} = t_{\text{submit}} - t_{\text{filled}} \tag{8}$$

I. Submission and Iteration

Once confirmed, the bot clicks the submit button and moves to the next filtered job posting.

Total time per application iteration:

$$T_{\text{iteration}} = T_{\text{login}} + T_{\text{scrape}} + T_{\text{relevance}} + T_{\text{fill}} + T_{\text{review}} \tag{9}$$

This metric helps in determining the efficiency of the system across different job portals.

J. Probability Metrics and Performance Evaluation

Success Probability

If feedback data is available:

$$P_{\text{success}} = \frac{\text{Number of Offers or Interviews}}{\text{Total Applications Submitted}} \tag{10}$$

Resume v/s Job Match Accuracy

Human-annotated relevance scores H can be compared to machine-predicted scores M to assess prediction quality:

$$\text{Accuracy}_{\text{match}} = \frac{|H \cap M|}{|H \cup M|} \tag{11}$$

K. Logging, Analytics, and OCR Comparison

Each job application process logs:

- Time Stamps: Start time, fill time, submit time.
- Relevance Score.
- Resume text source: OCR vs. Prompt.
- Application outcome (success/failure).

These logs enable the comparison of:

- OCR vs. prompt-based accuracy.
- Time efficiency across portals.
- Relevance prediction reliability.

IV. Results and Observations

Figure 1 illustrates the end-to-end pipeline of the automated internship application system powered by browser automation, Gemini 1.5 Flash LLM, and a custom relevance scoring engine. The workflow begins with resume upload (PDF or DOCX). If scanned, OCR is used for text extraction. Parsed text is stored in a prompt base, later used by Gemini to auto-generate form responses. User login credentials are securely stored in a `.env` file and injected via Selenium WebDriver for portal access. After manual CAPTCHA clearance, the bot scrapes job listings using HTML tag patterns to extract title, description, skills, and location. Each job is evaluated using a relevance score combining cosine similarity and keyword overlap. Only high-scoring jobs are shortlisted. Gemini then fills application forms based on parsed resume content. A manual review stage allows users to verify or modify responses before auto-submission. This loop repeats across relevant postings, enabling a streamlined, semi-autonomous application process.

To evaluate the system's performance, a comprehensive set of observations was recorded. The application workflow—ranging from resume parsing to job matching and form filling—was executed using a sample resume and tested across multiple job portals. Gemini 1.5 Flash was the primary LLM used due to its low latency, strong integration with Google tools, and support for structured outputs. The LLMs were benchmarked (qualitatively and partially quantitatively) across capabilities such as response time, contextual accuracy, structured form handling, and relevance scoring. **Resume Template Reference:** A sample resume used for testing relevance scoring and form filling is available at the following link: **Sample Resume link**

The jobs identified as relevant for this resume included:

- **Artificial Intelligence Intern at Jivichem Synthesis Private Limited**
- **Machine Learning Intern at Jivichem Synthesis Private Limited**
- **Data Science at DeepThought Edutech Ventures Private Limited**
- **AI Agent Developer at Ergode IT Services Private Limited**

Although full benchmarking across all LLMs was limited due to API access constraints, expected performance trends were visualized using public benchmarks and user feedback. Gemini 1.5 Flash demonstrated optimal performance in latency and structured task handling.

Observed Job Metrics for 'Research' Role:

```
Login time: 13.28s
Scrape time: 4.14s
```

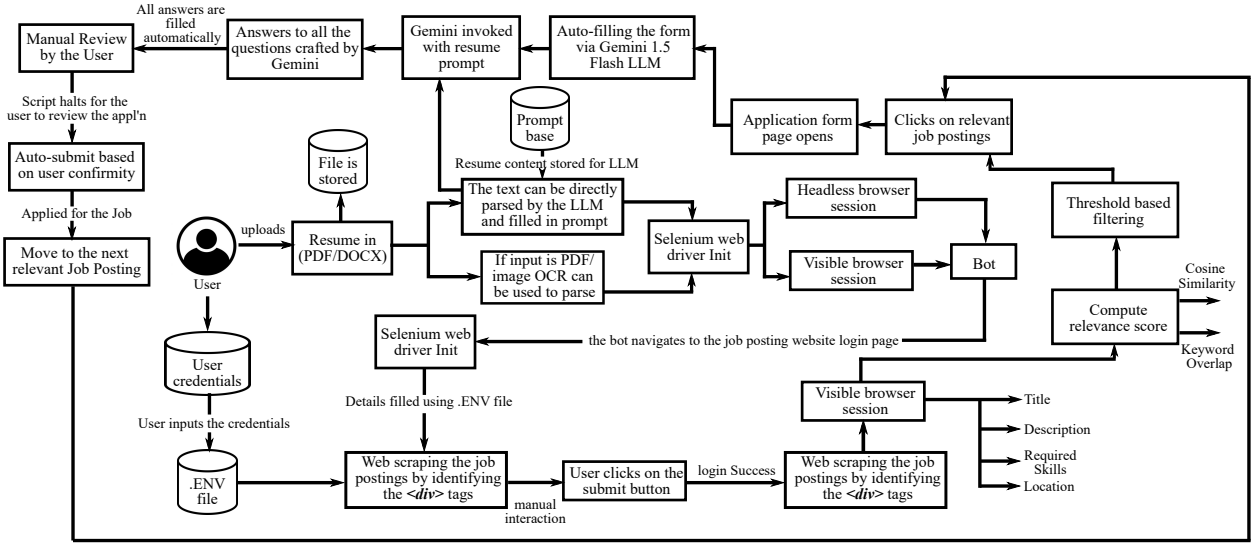

Fig. 1. End-to-end automated internship application system

TABLE I
COMPARISON OF LLMS ACROSS CAPABILITIES

Metric / Capability	**Gemini 1.5 Flash**	**GPT-4 (o4)**	**Claude 3 Opus**	**Mistral Large**	**Meta LLaMA 3**
Latency / Response Time	Very Fast (200 ms/token)	Moderate (300–500 ms/token)	Slower (500 ms+)	Fast	Fast
Form-Filling Suitability	Trained on instruction-following tasks, excels in short, structured text	Strong, slightly verbose	Good but cautious	Prone to underperformance in structured tasks	Still catching up in fine control
Context Window	Up to 1M tokens, easily handles large resumes + job descriptions	Up to 128k tokens	200k+ (Opus)	∼32k tokens	∼32k–65k (not as performant)
Inference Cost	Lower (if using via API, e.g., Vertex AI)	Higher (especially GPT-4 Turbo)	Higher	Efficient	Efficient
Relevance & Semantic Matching	Strong embedding and retrieval when fine-tuned	Excellent with embeddings (especially OpenAI)	Great on nuance	Weaker contextual understanding	Less robust on resume-like formats

```
Relevance scoring time: 0.52s
Fill-time(cover letter+questions):24.48s
Review/Submit time: 11.14s
Total iteration time: 42.35s
```

Figure 3 shows a clear efficiency advantage in using LLM-assisted automation over manual application workflows. The reduction in iteration time per job was substantial, improving scalability and throughput for internship applications.

V. CONCLUSION

This paper presents a modular and scalable methodology for automating internship applications by integrating Selenium-based web automation with Large Language Models (LLMs) such as Gemini-1.5-Flash. The system effectively automates key stages including resume parsing, job posting extraction, relevance evaluation using NLP-driven similarity measures, and context-aware form population. By combining traditional automation

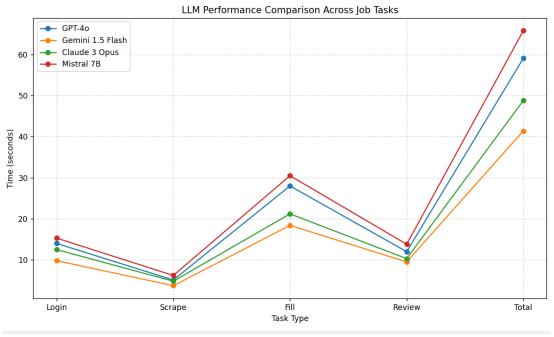

Fig. 2. Comparison of different LLMs (GPT-4o, Gemini 1.5 Flash, Claude 3 Opus, Mistral 7B) across various job automation metrics such as login time, scraping, fill time, and review time.

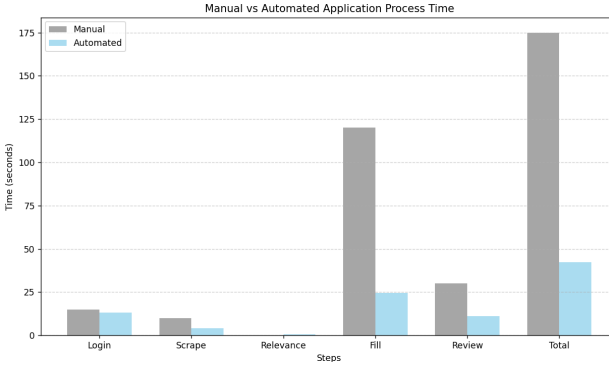

Fig. 3. Comparison of time taken to complete job application tasks manually versus using an automated system powered by LLMs. The automated pipeline significantly reduces time spent on repetitive tasks such as logging in, scraping job details, filling cover letters and forms, and reviewing submissions.

techniques with state-of-the-art LLM capabilities, the framework significantly reduces application time, improves consistency, and enhances the decision-making process in internship application workflows. Despite its advantages, the system encounters several limitations. Manual intervention remains necessary in cases involving CAPTCHA challenges or multifactor authentication, limiting complete automation. The performance of OCR is sensitive to document quality, which may impact text extraction accuracy. Furthermore, while the relevance scoring mechanism performs well for general use cases, it may require domain-specific tuning to maintain accuracy across varied job descriptions. The generative responses produced by the LLM, although contextually aligned, may occasionally lack specificity or exhibit redundancy, necessitating user review. Future enhancements may involve integrating CAPTCHA solvers, improving OCR robustness, and incorporating user feedback loops to fine tune relevance scoring and response generation.

REFERENCES

[1] Ruchitaa Raj NR, M Vijayalakshmi, et al. Web scrapping tools and techniques: A brief survey. In *2023 4th International Conference on Innovative Trends in Information Technology (ICITIIT)*, pages 1–4. IEEE, 2023.

[2] Ajay Sudhir Bale, Naveen Ghorpade, S Rohith, S Kamalesh, R Rohith, and BS Rohan. Web scraping approaches and their performance on modern websites. In *2022 3rd International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pages 956–959. IEEE, 2022.

[3] YS Shriya, Aarav Babu, Naren Chandrashekhar, Siddhant Jagdish Verma, and PT Shanthala. Automation of data analysis and web-scraping to value used items. In *2024 3rd International Conference on Artificial Intelligence For Internet of Things (AIIoT)*, pages 1–6. IEEE, 2024.

[4] Inzali Naing, Nobuo Funabiki, Khaing Hsu Wai, and Soe Thandar Aung. A design of automatic reference paper collection system using selenium and bert model. In *2023 IEEE 12th Global Conference on Consumer Electronics (GCCE)*, pages 267–268. IEEE, 2023.

[5] Mutaz Abdel Wahed, Mowafaq Salem Alzboon, Muhyeeddin Alqaraleh, Jaradat Ayman, Mohammad Al-Batah, and Ahmad Fuad Bader. Automating web data collection: Challenges, solutions, and python-based strategies for effective web scraping. In *2024 7th International Conference on Internet Applications, Protocols, and Services (NETAPPS)*, pages 1–6. IEEE, 2024.

[6] Bhavesh Pandekar and Savita Sangam. Build near real time social media intelligence using web scraping and visualization. In *International Conference on Intelligent Computing and Networking*, pages 493–504. Springer, 2023.

[7] Ayat Abodayeh, Reem Hejazi, Ward Najjar, Leena Shihadeh, and Rabia Latif. Web scraping for data analytics: A beautifulsoup implementation. In *2023 Sixth International Conference of Women in Data Science at Prince Sultan University (WiDS PSU)*, pages 65–69. IEEE, 2023.

[8] Arief Agus Sukmandhani, Thefan Sunjaya, Immanuela Puspasari Saputro, and Jenny Ohliati. Data scraping using python for information retrieval on e-commerce with brand keyword. In *2023 8th International Conference on Business and Industrial Research (ICBIR)*, pages 179–183. IEEE, 2023.

[9] Adithyan Sasikumar T, Sandeep Singh Chauhan, and Sarda Sharma. Automated web scraping for telecom corpus application. In *2024 IEEE 3rd International Conference on Data, Decision and Systems (ICDDS)*, pages 1–5. IEEE, 2024.

[10] Rishabh Bhargava, Russel Lobo, Rushabh Shah, Nishank Shah, and Sindhu Nair. Easier web navigation using intent classification, web scraping and nlp approaches. In *2022 5th International Conference on Advances in Science and Technology (ICAST)*, pages 286–290. IEEE, 2022.