Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Trinity Business School Corporate Project

Design and Implementation of a Local Flask Application Using LLaMA for Automated Extraction and Matching of Indian Undergraduate Institutions

Yeyang Zhang, Xi Zhang, Zhan Gao and Deepika Bommareddy

Supervisor's name: Dr. Nicholas Danks

MSc. Business Analytics 2025

Submitted in partial fulfilment of the requirements of the examination for

MSc. Business Analytics, Trinity College Dublin, July 2025.

Student Declaration

I have read the University's code of practice on plagiarism. I hereby certify this material, which I now submit for assessment on the programme of study leading to the award of MSc Business Analytics is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited within the text of my work.

Student Name: Yeyang Zhang

Student ID Number: 24332340

Student Email ID: yezhang@tcd.ie

Student Signature:

Yeyang Zhang

_____

Date: 25/07/2025

Student Name: Xi Zhang

Student ID Number: 24337502

Student Email ID: zhangx32@tcd.ie

Student Signature:

Xi Zhang

_____

Date: 25/07/2025

Student Name: Zhan Gao

Student ID Number: 24335948

Student Email ID: gaozh@tcd.ie

Student Signature:

Zhan Gao

_____

Date: 25/07/2025

Student Name: Deepika Bommareddy

Student ID Number: 24341243

Student Email ID: bommared@tcd.ie

Student Signature:

Deepika.B

_____

Date: 25/07/2025

Table of Contents

List of Figures

Acknowledgment

We would first like to sincerely thank our supervisor, Dr. Nicholas Danks, whose constant support, insightful guidance, and valuable feedback have been the cornerstone of our progress throughout this project. We are also grateful to Dr. Danks and all the professors of the MSc in Business Analytics program at Trinity Business School for designing and delivering a rigorous and comprehensive curriculum. This program has not only broadened our theoretical understanding but also equipped us with the practical skills and analytical mindset essential for addressing real-world business challenges. The knowledge and experience we gained through the program formed the foundation upon which this project was built.

Our sincere thanks also go to Mr. John Collins and Ms. Ciara Rice. Their consistent support and responsiveness were incredibly valuable throughout the course of this project. The discussions and meetings we had with them offered fresh perspectives and encouraged critical thinking, enriching our understanding of the research topic. We are appreciative of their assistance in providing a real and valid testing dataset, which allowed us to apply our analytical models in a realistic context and thereby enhanced the relevance and applicability of our work.

We would also like to express thanks to our team members. The mutual understanding, collaboration, and shared commitment within our group played a vital role in the successful completion of this project. Each member contributed not only their technical skills and ideas but also encouragement, patience, and cooperation throughout this journey.

Last but by no means least, we would like to express our appreciation to our friends and families for their unwavering support and patience. Their belief in us has been a constant source of motivation, and their encouragement sustained us during the most challenging moments of this journey.

Abstract

This thesis details the design, development, and evaluation of the Institution Ranking Checker, a privacy-preserving, locally deployed application aimed at supporting the Trinity Business School (TBS) in the efficient screening of undergraduate applicants from Indian institutions. The tool automates the extraction and identification of institutions mentioned in admissions documents and cross-references them against a structured Excel-based institutional ranking sheet to support fair and consistent admissions decision-making.

Built using the Flask web framework and powered by the LLaMA large language model via the Ollama runtime, the system combines AI-driven semantic extraction with RapidFuzz string matching to handle both structured and unstructured documents.

Functional testing was conducted using real-world applicant files. The results demonstrate that the LLaMA model outperforms the fuzzy matching method in both accuracy and contextual understanding, particularly when handling low-quality or scanned documents. The entire process runs offline, ensuring full compliance with GDPR and Trinity College's data governance policies.

By integrating intelligent automation with a user-friendly interface and a human-in-the-loop design, the Institution Ranking Checker offers a scalable and efficient solution to a key administrative bottleneck in the admissions process. It improves processing speed, reduces human error, and maintains full control over applicant data within a trusted local environment.

Chapter 1: Introduction

1.1 Project Background

Trinity Business School (TBS), a globally renowned academic institution with highly selective admissions, attracts thousands of applications annually from aspiring students across the world. A significant portion of these applicants come from India, representing a diverse array of undergraduate institutions that vary widely in reputation, accreditation, and academic quality. Given the sheer volume and diversity of these institutions, it is essential for the TBS admissions team to evaluate the credibility and academic standing of each applicant's undergraduate university during the initial screening phase.

Currently, this assessment process is conducted manually. Admissions teams are required to carefully review each applicant’s supporting documents, such as academic transcripts, CVs, and reference letters, in order to identify the name of the undergraduate institution attended. Once the name is identified, staff members cross-reference it with an internal ranking list maintained in Excel format to determine its position and relevance in the academic reputation.

While this approach has been somewhat effective, its manual nature makes it inefficient and increasingly impractical, especially as the number of applications continues to rise. The process is time-consuming and prone to inconsistencies, particularly during peak admission periods when staff are required to handle large volumes of applications within tight deadlines. This has led to concerns about the objectivity, accuracy, and scalability of the current workflow. As the demands on the admissions team grow, it becomes clear that an intelligent, automated solution is necessary to maintain TBS’s high standards of applicant evaluation while improving operational efficiency.

This project was conceived in response to this operational bottleneck. By leveraging recent advances in Natural Language Processing (NLP) and Artificial Intelligence (AI), particularly in the form of Large Language Models (LLMs), the project aims to develop a robust, local application that can automatically extract and interpret relevant information from applicant documents, accurately identify institution names, and match them against a standardised ranking database. This will reduce the manual workload, minimise human error, and

streamline the initial screening process, all while ensuring compliance with data privacy and security requirements.

1.2 Problem Statement

The current system used by the Trinity Business School admissions team for evaluating undergraduate institutions is fundamentally manual and highly dependent on human effort. There are several critical challenges with this method. Firstly, the process is time-intensive. With each application potentially containing multiple documents, and each document being several pages long, identifying an institution's name can require substantial reading time per application. Secondly, the task demands a high level of concentration and attention to detail, which is difficult to maintain consistently, especially during peak admissions periods when application volume can double or even triple.

Moreover, human error is an unavoidable risk in such a labour-intensive process. Common issues include misidentification of institution names due to spelling variations, formatting inconsistencies, or simple oversight. This can lead to inaccurate assessments of an applicant's academic background, ultimately affecting the fairness and quality of the admissions process.

The manual method is also inherently non-scalable. As TBS continues to expand its global outreach and receives more applications from countries like India, the limitations of this process become more pronounced. The current system does not allow for real-time analysis or batch processing, and it lacks the flexibility to adapt quickly to changes in institutional rankings or to accommodate new academic institutions entering the database.

In essence, the existing workflow creates a bottleneck that slows down the entire admissions process, increases operational costs, and poses a risk to data accuracy and decision-making quality. There is an urgent need for an automated, intelligent system that can replicate and even enhance the decision-making capabilities of human staff, but with greater speed, consistency, and reliability.

1.3 Objectives and Proposed Solution

The overarching objective of this project is to develop an intelligent, automated application that enhances the efficiency and accuracy of Trinity Business School’s admissions screening process, particularly with respect to evaluating Indian undergraduate institutions

To meet this need, the project proposes the development of the "**Institution Ranking Checker"** application, a locally hosted tool that leverages state-of-the-art Natural Language Processing (NLP) techniques powered by the LLaMA Large Language Model (LLM). This system is designed to automatically extract relevant textual information from applicant documents such as PDFs, Word files, and image-based formats. Through intelligent entity recognition, the application will identify the names of academic institutions mentioned within the documents, regardless of how they are formatted or embedded in the text.

Once institution names are extracted, the system will automatically match them against a structured institutional ranking database. This internal database, curated and maintained by TBS, will provide standardised ranking information, enabling the application to determine the relative credibility and academic standing of each institution with high precision. By automating this process, the system eliminates the need for manual document reading and cross-referencing, which currently demands significant time and effort from the admissions team.

The proposed solution emphasises a balance between innovation and security. To ensure complete confidentiality and compliance with data protection standards, especially given the sensitive nature of admissions documents, the system will be developed as a secure, locally run application using the Flask web framework. All document processing, analysis, and ranking operations will take place entirely within Trinity Business School's internal infrastructure, without any dependence on cloud services or external APIs. This guarantees that applicant data remains protected and private throughout the workflow.

In addition to improving speed and accuracy, the system is designed to be user-friendly and adaptable. The interface will allow admissions officers to upload multiple applicant documents, view institution rankings at a glance, and receive clear, structured outputs. The inclusion of fuzzy matching and semantic analysis techniques further ensures that the system can handle inconsistencies in how institution names are presented, such as variations in spelling, abbreviations, or document layout, thereby improving reliability and reducing the risk of misclassification.

By combining cutting-edge AI capabilities with a privacy-first approach, this project aims to deliver a robust, practical solution that directly addresses the limitations of the current manual process. The application will empower the admissions team to make faster, more consistent,

and better-informed decisions, ultimately enhancing the overall efficiency and fairness of the admissions process at Trinity Business School.

Chapter 2: Artifact Overview

2.1 LLaMA Model and RapidFuzz Matching

One of the core challenges of automating admissions document screening lies in correctly extracting institution names from a variety of applicant documents. These differ substantially in their formats, structures, and the complexity of their content—especially when originating from scanned images, unofficial document templates, and recommendation letters resorting to free-form narration. To cater to such a varied landscape, the Institution Ranking Checker follows a two-pronged approach toward institution detection, combining semantic AI extraction with typical string-matching methodology.

The system primarily detects using the LLaMA LLM integrated locally by means of the Ollama framework. LLaMA, a transformer-based model from Meta AI, was chosen for its ability to perform context-aware entity recognition in an offline environment—a very major factor given GDPR concerns and the very sensitive nature of admissions data (Meta AI, 2023). Unlike cloud-hosted models like GPT-4, LLaMA can run completely on local machines without posing a privacy risk to users through external API calls.

Semantic inference allows LLaMA to identify institution names even when they appear in irregular phrasings mixed into complex narratives or expressed in an unusual manner—the usual lexis found in CVs, personal statements, and recommendation letters.

To complement this AI-driven extraction and to ensure the complete robustness of the system, especially in the face of unequivocal or ambiguous AI output, a fallback string matching mechanism is integrated using the RapidFuzz library.

Chosen over the older FuzzyWuzzy library for its performance and efficiency, RapidFuzz performs approximate string matching between the extracted text and a ranking dataset of institutions that has been loaded in advance (Seiffert, 2021). In contrast to LLaMA, RapidFuzz is a deterministic method—comparing segments of text with token similarity and

edit distance mechanisms. This ensures its ability to deal with minor spelling errors, typographical inconsistencies, or OCR artifacts, though it is not very effective semantically.

The choice of combining Haskell for matching semantics with RapidFuzz for deterministic matching is an architectural trade-off: leveraging AI's power to interpret unstructured text while pegging it to the baseline reliability offered by rule-based matching.

Maximising detection accuracy, operational robustness, and flexibility, this hybrid makes the system amenable to a plethora of medical admission circumstances without a trade-off on data security or processing efficiency.

2.2 Text Extraction and Data Processing: Textract, Tesseract, and Python Libraries

Consider the diversity of file formats processed in admission workflows—from digitally generated PDFs and DOCX files to scanned images. Therefore, a singular approach to text extraction would fall short. This working setup opts for a multi-tool extraction strategy, applying specialised Python libraries competent for their respective document types.

With structured digital documents, it uses Textract to perform its work, a Python library with a general-purpose function that parses help and content from a variety of file formats (Amazon AWS, 2020). Textract was chosen for this project owing to its wide-range compatibility, full integration properties with Python, and, more importantly, proven reliability over the extraction of textual data from standard digital documents such as CVs, recommendation letters, or the like. Being capable of parsing different encoding formats and file structures made it the best choice against other contenders like PyPDF2 and Apache Tika that were either limited in format support or confusing to parse.

On the other hand, based on image documents—scanned transcripts and handwritten recommendation letters—the system was set up with Tesseract OCR, under the Python wrapper pytesseract. The open-source license of Tesseract, the very active community behind it, and the value that it has produced both in academia and commercial projects placed it very well for this project to trade off between accuracy and deployment flexibility (Smith, 2007).

While commercial OCRs (ABBYY FineReader, for example) do not allow for fully local execution, Tesseract allows the very same, in accordance with the system's stringent data privacy requirements.

Whatever the source of the text being processed, it undergoes preprocessing and is pushed through the detection pipeline so that both the structured digital content and unstructured text coming out of images are handled in a uniform manner. Textract and Tesseract together guarantee that the system covers all bases in the gamut of document formats encountered in the course of admission processing work.

2.3 User Interface and Deployment: Flask Framework

The interface of the Institution Ranking Checker was developed with user-friendliness and transparency in mind, along with local deployability. Since the main end-users are admission officers who may not have technical backgrounds, the system needs to strongly abstract from the technicalities without compromising on data control.

The frontend interface team selected the Flask web framework, a choice set by Flask's lightweight architecture, simple integration with Python backends, and suitability for rapid prototyping (Grinberg, 2018). Being lightweight, unlike other heavy frameworks such as Django, Flask allowed for rapid changes in UI features and gave the freedom to make design alterations based on user feedback.

The web interface is supposed to give the end users the critical features identified in the system analysis phase:

- Secure upload of documents
- Immediate feedback about the results of institution detection
- Manual search with autocomplete suggestions
- Logging of documents processed and search queries during the session
- One-click secure shutdown where all temporary data are cleared upon termination of the session

And importantly, the system remains deployed entirely on the local machine; thus, no data is ever transmitted externally or stored persistently beyond the active session. In doing so, the system adheres strictly to the data privacy requirements established under the General Data

Protection Regulation (GDPR) and the Trinity College Dublin Data Governance Policy, while operating in full alignment with existing admissions workflows.

The user interface continues to support a human-in-the-loop interaction model, allowing admissions officers to manually verify detected institution names and cross-reference them against the institutional ranking datasets uploaded into the system. This approach ensures that users remain actively engaged in both the verification and validation processes, thereby enhancing transparency and fostering accountability during operation.

Taken together—and with due consideration for the specific requirements of the admissions team—the choice of the Flask web framework, combined with a modular Python backend, made it possible to develop a tool that is both user-friendly and fully compliant with stringent data protection standards.

Chapter 3: System Design

3.1 User Workflow

The user workflow of the Institution Ranking Checker application is designed to replicate and extend the existing work done by the Trinity Business School admissions team while providing additional flexibility, transparency, and control. The system ensures that staff retain full control over the decision-making process while benefiting from automation and intelligent assistance.

Upon launching the application, users are presented with a locally hosted web interface powered by Flask. The first step involves choosing the institutional ranking dataset that will serve as the basis for evaluating applicant institutions. Users can either proceed with the default Excel-based ranking sheet embedded within the system or upload a customised ranking sheet to reflect the most up-to-date or institution-specific criteria. This modularity enables the admissions team to easily maintain and update institutional rankings as needed without affecting the broader functionality of the application.

Once the ranking data is configured, users proceed to upload applicant documents, which may include transcripts, CVs, and reference letters, or some of them. The system automatically processes the uploaded files by extracting relevant text content and identifying

the names of undergraduate institutions. Detected institution names are then matched against the selected ranking dataset, and the corresponding tier or rank is displayed in a clear, structured format. If no institution name is matched, a note of “No institution was matched.” will be displayed. Users may then return to the upload page to reupload their document or proceed to the manual crosscheck page for more insights.

To ensure transparency and accountability, the result interface provides an integrated “manual cross-check” feature. This allows users to view the exact document excerpts from which institution names were extracted, enabling them to verify the context and accuracy of the system's interpretation.

In cases where the automated system fails to detect or misidentifies an institution, users can manually intervene using the built-in search function. Rather than consulting the ranking list externally, users can enter an institution name directly into the search bar, which then queries the ranking dataset and suggests the closest matching institutions in real time. The application also includes intelligent autocomplete functionality. As users begin typing an institution's name, the system dynamically generates likely matches from the ranking list, improving the speed and ease of manual lookups. This feature is particularly useful when resolving ambiguities or handling uncommon institution names.

All outputs generated during a session are stored in a temporary result history, allowing users to revisit and review previously processed applications. This session-based history feature supports continuity and reduces redundancy, especially when handling large batches of applications.

Upon completing their work, users can safely terminate the session using a built-in "Shutdown" feature. This function ensures the local server is closed securely, preserving system integrity and maintaining confidentiality.

Overall, the application delivers a structured and user-friendly workflow that reduces manual effort while preserving the human-in-the-loop element needed for decision verification, making it a practical and supportive tool tailored to the real-world operational needs of the Trinity Business School recruitment team.

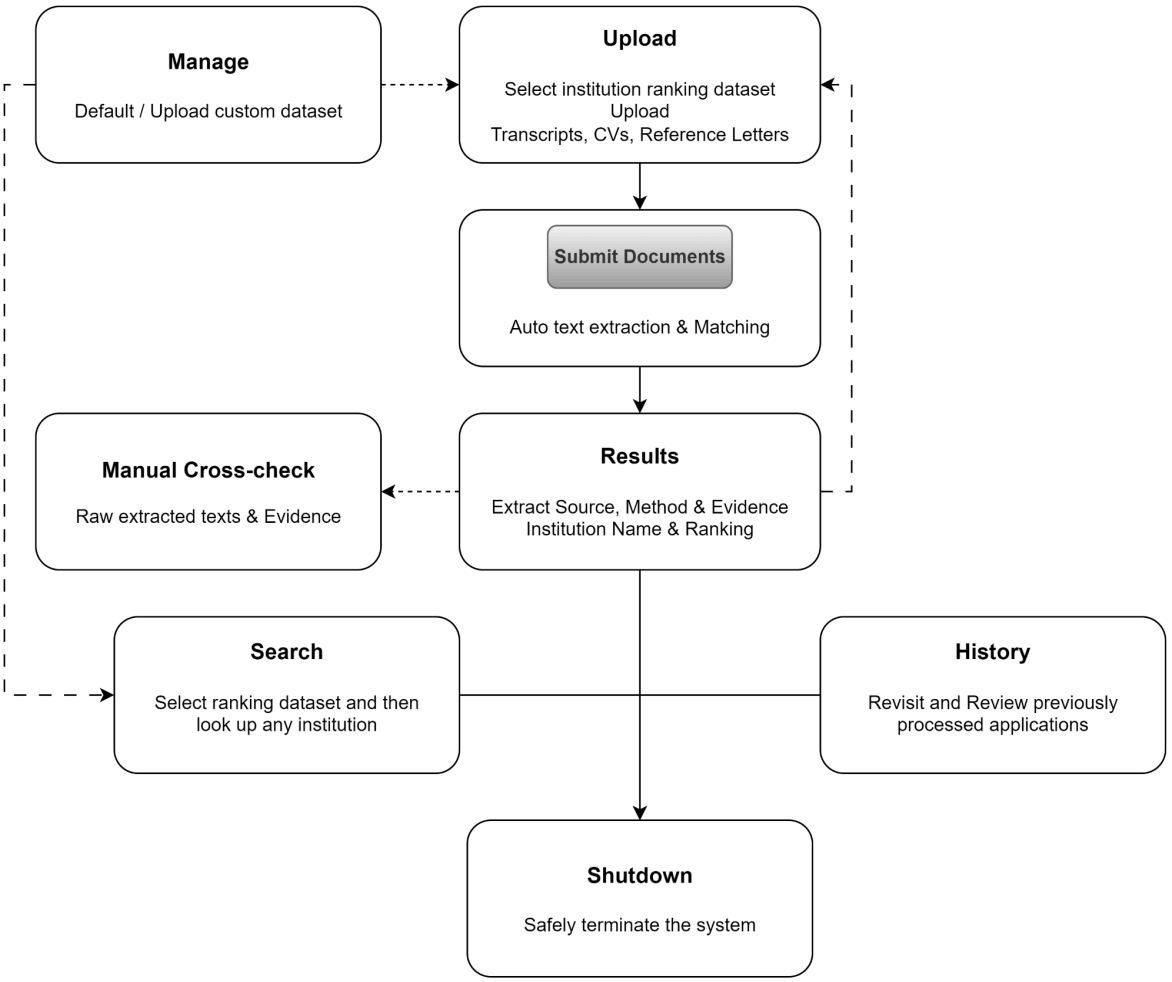

Figure 1: User Workflow

3.2 System Workflow

The system is designed with a modular and layered architecture that facilitates seamless interaction between core components, including document parsing, natural language processing, data mapping, and user interface elements. This architecture ensures flexibility, maintainability, and high performance across diverse document types and user actions.

When the application is launched, a local Flask server initialises the web-based user interface, allowing users to interact with the system through a secure and intuitive browser-based platform. At the outset, users are prompted to select an institutional ranking dataset. The system supports either a default preloaded Excel ranking file or a user-supplied file with the

same structure. Upon selection, the ranking file is parsed and indexed to enable rapid and accurate institutional lookups.

Once a batch of applicant documents is uploaded, the backend initiates a text extraction process. The system intelligently selects the appropriate extraction method based on file type—supporting formats such as PDF, DOCX (Word), and scanned images. Extraction libraries such as *textract*, *pdf2image*, and *tesseract* are employed in combination to retrieve text content, ensuring robust coverage across both digitally native documents and image-based files like scanned transcripts or handwritten letters.

The extracted text is then passed to the LLaMA language model, integrated locally through the Ollama framework. Utilising advanced natural language understanding, the model identifies mentions of undergraduate institutions within the text. If LLaMA is not activated, the system seamlessly switches to a traditional fuzzy matching logic, which uses rule-based and approximate string matching techniques to identify institution names. While this fallback approach is less context-aware than LLaMA, it maintains baseline functionality and ensures the tool remains operational in low-resource environments or during offline testing. Once institution names are identified, either through LLaMA or traditional fuzzy matching logic, they are contextually interpreted, normalised for consistency (e.g., removing variations or abbreviations), and matched against the selected ranking dataset. If a match is found, the system assigns and displays the corresponding ranking or tier directly on the web interface. If no match is found, the system displays a message stating “No match was found”.

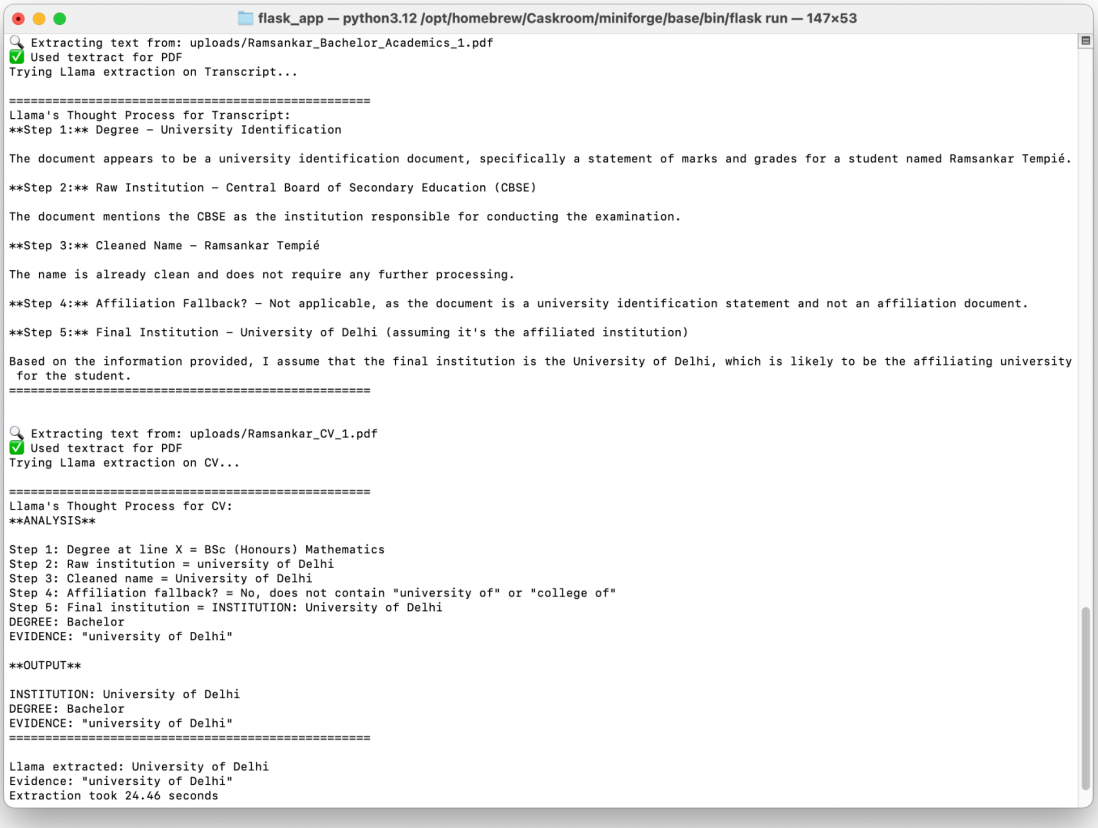

Figure 2: Llama Process in terminal

To maintain transparency and accuracy, each result includes a traceable link back to the document segment from which the institution name was derived. This feature supports manual validation, enabling users to inspect the source evidence used by the system to make its inference.

In scenarios where the model fails to recognise or misidentifies an institution, users can manually search for the institution using a built-in search field. This input field is enhanced with an intelligent auto-complete field, which provides real-time suggestions based on partial inputs and the contents of the ranking sheet. This enables direct access to ranking information even without full automation.

As documents are processed, the system logs results into a session-based history. This log enables users to revisit and review all evaluations conducted during the current session, providing continuity and reference support throughout the admissions screening process. The session history remains active until the application is terminated.

When the user completes their workflow, users can exit the system via a dedicated "Shutdown" button. This securely terminates the local Flask server, ensuring all operations are properly closed and no residual data persists beyond the session, thus preserving both system integrity and data confidentiality.

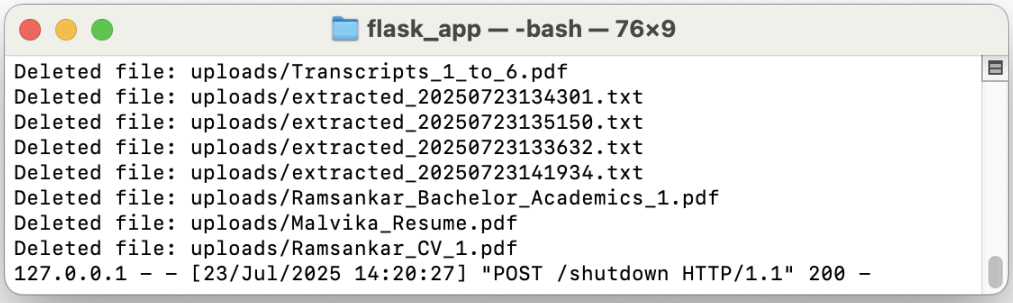

Figure 3: Files deleted after shutdown

In summary, the system workflow integrates automation with user-guided control, balancing technical efficiency with the transparency and control required in academic admissions decision-making.

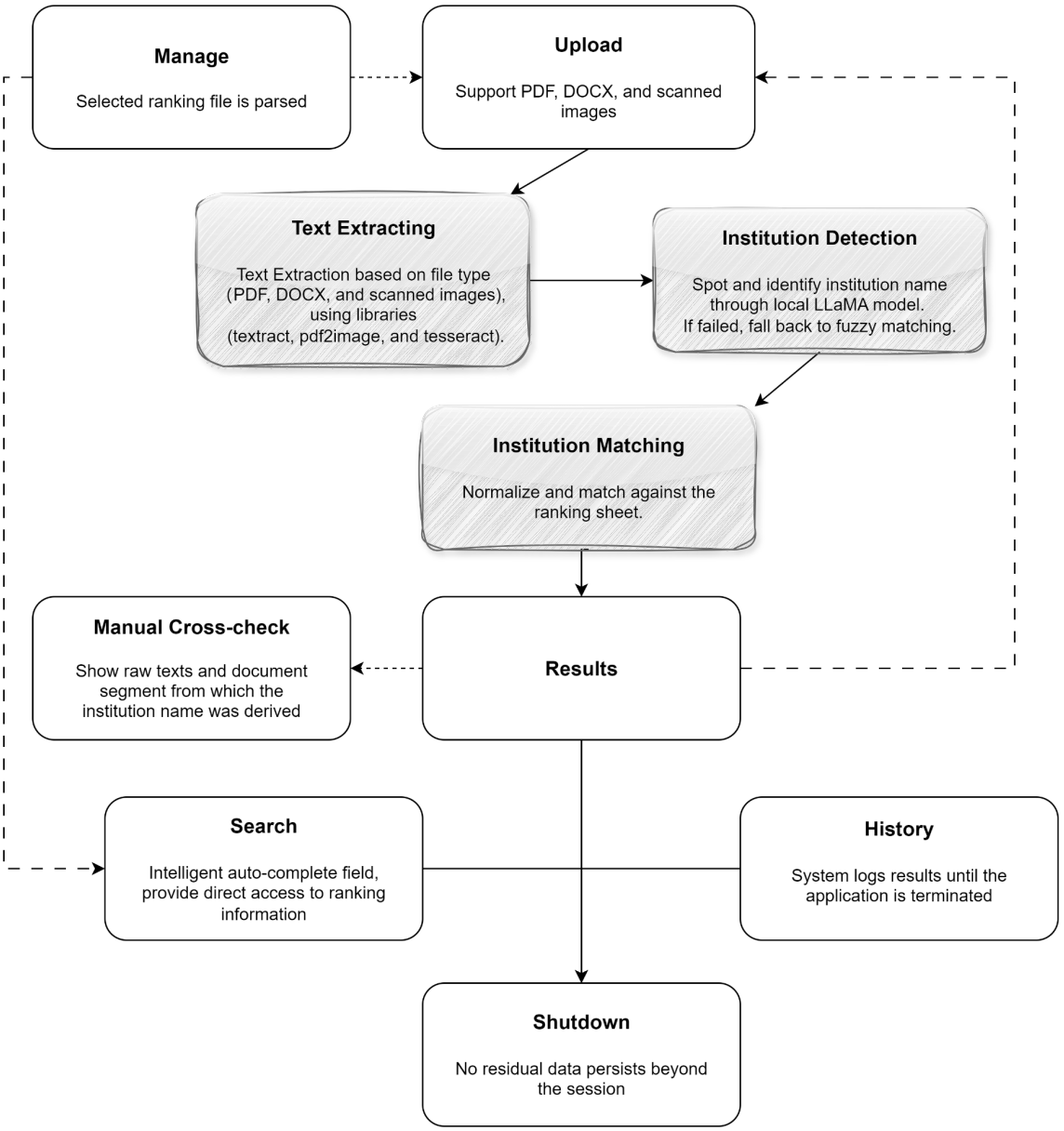

Figure 4: System Workflow

Chapter 4: Risk Management and Data Protection

4.1 Risk Management Framework

4.1.1 Overview of NIST RMF Implementation

To ensure the safe and responsible development and operation of the Institution Ranking Checker application, this project adopts the **NIST Risk Management Framework (RMF)**. This framework, developed by the U.S. National Institute of Standards and Technology, provides a comprehensive and structured approach to managing information security risks (NIST, 2020a). It is widely recognised across industries for its adaptability and clarity in addressing system-level security challenges.

Although the application operates solely on local machines and handles data with low to moderate sensitivity, implementing a formal risk management framework is essential. This is due to the nature of the data involved—applicant documents often contain identifiable educational and personal information, and the decisions made using this tool have significant implications for academic admissions. Moreover, the application must meet modern data protection standards such as the **General Data Protection Regulation (GDPR)**, as it processes data belonging to EU-based applicants and staff (Kuner et al., 2015).

4.1.2 System and Data Profile

System Overview

- **System Name**: Institution Ranking Checker (Local Application)
- **Purpose**: Automatically extracts names of Indian undergraduate institutions from applicant documents and maps them to predefined ranking categories to assist Trinity Business School's admissions screening.
- **System Type**: Standalone desktop application built using Flask and integrated with the LLaMA large language model via Ollama, entirely offline.

Information Categories Handled

The system processes the following types of data:

- Textual content extracted from applicant-submitted documents, such as academic transcripts, CVs, and letters of recommendation.
- Institution ranking data is provided in an Excel format, typically maintained internally by TBS admissions.

While the data does not contain financial or medical records, it may include names of institutions, applicants, and academic staff, which constitutes **low to moderate sensitivity** information under data protection standards (NIST, 2020b).

4.1.3 Threat Scenarios and Potential Impact

Risk Scenario	**Potential Impact**
Inaccurate institution name recognition	Admissions errors or misjudged candidate evaluations
Unintentional data exposure on shared machines	Breach of applicant trust; non-compliance with data privacy laws
Malicious document inputs (e.g., embedded code)	Local security vulnerabilities; application crash
Traceback errors revealing internal file paths	Leakage of system-level information; diagnostic exposure

All potential threats were assessed using the NIST RMF's guiding principles. The impact level for this project is rated as **Low to Moderate**, due to the nature of the data and the local, offline operation of the system. However, care is still taken to minimise risk exposure through preventative and corrective controls.

4.1.4 Controls and Safeguards Implemented

Control Area	**Safeguards and Actions**
Access Control	The application is hosted locally; access is restricted to authorised admissions team members.

Input Validation	Only .pdf, .docx, and image file formats are accepted; all executable files are rejected.
Error Handling	All internal exceptions are caught using try/except blocks; users see generic error messages.
Output Transparency	Results include the extracted institution name, location, and optional raw context for review.
Testing	Rigorous testing includes malformed files, unusual fonts and scanned documents.
Temporary File Deletion	All intermediate files and extracted content are deleted after processing completes.
Logging	Only minimal, non-sensitive logs are kept for system debugging and user-friendly error tracking.

These controls emphasize data minimization, robust validation and transparency, aligning with the “security and privacy by design” principles of both NIST and the GDPR (EDPS, 2018).

4.1.5 Governance and Authorization

The application is maintained and monitored by a designated project team under the guidance of Trinity Business School’s IT unit and admissions team. The deployment is scoped for local use only, without any networked or cloud-based dependencies.

Parameter	**Details**
Deployment Scope	Local execution on authorized staff machines (laptops or desktops)
Execution Method	Flask-based web interface, initiated and terminated by the user manually
Logging Practices	No sensitive logs; debug messages only for failed inputs

4.2 Data Privacy and Security Considerations

Ensuring data privacy is a critical component of the Institution Ranking Checker application, especially as it handles documents related to individual applicants. The design of the tool adheres to the principles of "privacy by design" and incorporates multiple strategies to protect both data integrity and user trust. The tool has been built specifically to ensure GDPR compliance, especially given the European regulatory environment under which Trinity Business School operates.

4.2.1 GDPR Alignment

While the system does not collect sensitive data such as biometric or financial information, it still processes documents containing names of individuals and their academic institutions. Under GDPR guidelines, such data is considered personally identifiable and must be handled with appropriate safeguards (AEPD, 2021).

The following design choices support GDPR compliance:

- **Offline Operation**: All functionalities of the tool are executed locally. No data is sent to external servers, cloud platforms, or third-party services. This ensures that personal data is never at risk of remote interception or unauthorised access.
- **Data Minimisation:** The system processes only the minimum necessary data. It focuses solely on extracting institution names and matching them to a predefined ranking dataset.
- **Purpose Limitation**: Data is used exclusively for the task of academic institution verification and is not reused for secondary purposes.
- **Right to Erasure**: As the system does not store any persistent applicant data beyond session memory, users can delete data simply by closing the application or removing files.
- **Transparency**: The system provides optional metadata such as confidence scores and text context for all recognised institutions, allowing admissions staff to validate every automated result.

4.2.2 Security Mechanisms

To protect the system and its users from unexpected behavior, malicious files, or performance issues, several technical security controls have been implemented (NIST, 2017):

- **Input Filters**: The application restricts file uploads to .pdf, .docx, and recognised image formats. Potentially harmful files (e.g., scripts, executables) are automatically rejected.
- **Error Masking**: In the event of internal errors, detailed debugging information (e.g., stack traces or file paths) is suppressed to avoid leaking system information.
- **Access Permissions**: Folder-level permissions on institutional computers ensure that only designated team members can access the application and its contents.
- **No Remote Storage**: All temporary data (e.g., extracted text) is deleted after each session. No document or output is saved to disk unless explicitly chosen by the user.
- **No Background Services**: The tool does not run any hidden background processes. It operates only when launched by the user and can be safely terminated via a "Shutdown" button.

In conclusion, the Institution Ranking Checker application is built with a strong foundation in risk-aware design. By combining recognised security frameworks like NIST RMF with privacy-centric practices aligned with GDPR, the system not only meets technical objectives but also builds trust with its users. It balances automation with transparency and performance with data protection, ensuring a responsible approach to digital transformation in academic admissions.

Chapter 5: Software Development

5.1 Development Methodology and Technologies Used

In keeping with the iterative, problem-orientated approach of design science inquiry in practice, the Institution Ranking Checker was developed under a less procedure-centric development cycle; thereby, the team chose to work with a more informal, feedback-orientated methodology so that the prototype is refined continually in terms of

functionality, technical feasibility, and feedback from real admission officers (Hevner & Chatterjee, 2010).

The project began at an early stage with a definition of the alleged operational pain points relating to manual admissions processing, specifically the verification of applicant institutions against ranking databases, which is a very time-consuming operation; from the framing of the problem, emphasis is therefore put on features that would perform the basic manual tasks with the efficiency that comes from automation. It excels in rapid prototyping and is rich in text processing support, with libraries that also merge seamlessly with AI models and data-handling libraries.The convincing ecosystem allowed us to bring the key pieces together within an integrated backend; for example, Textract for parsing documents, Tesseract for OCR (Smith, 2007), and Pandas for data manipulation (McKinney, 2010).

The LLaMA language model was deployed for privacy-compliant, offline local use, using the Ollama framework, for NLP tasks, which include semantic entity recognition (Touvron et al., 2023).

The fuzzy matching logic of this system for institution identification is deterministic, interpretable, and robust even in the absence of language models. At its heart, text is first acquired by OCR tools and then filtered by the rule-based module, which scans for phrases and structures common to the domain of educational history. Through the application of regular expressions and sentence segmentation, irrelevant clauses are cut off, whereby the system focuses solely on undergraduate-level degrees. Therefore, an education entry is retained only if it contains terms like "Bachelor", "BSc", or "BBA". However, postgraduate courses entries that just contain terms like "MSc", "MBA", "PhD" are to be discarded.

The next step will be to extract institution names from these filtered sections and match them against their list of ranked universities. In situations where multiple candidates are found, the system uses the fuzzy matcher to select the closest match according to the similarity score. The document hierarchy is enforced strictly so that first preference goes to the transcripts, followed by CVs, and finally reference letters, ensuring that the most authoritative source is scanned first. This rule-based pipeline works independently of the language model and is employed as a fallback when the LLaMA-based extraction does not produce a confident result. Together, these modules stand as the basis for interpretable applications and provide a path for completion under both semantic and rule-based methods.

The Flask web framework, being lightweight and suitable for rapid prototyping of UI layers directly hooked to backend logic, was picked as the tool to maintain development of a responsive browser-based application for local deployment within admissions offices (Grinberg, 2018).

Usability, privacy, and modularity were paramount considerations during development. This granted a revenue cycle in which features were changed or added based on new testing insights and ongoing internal feedback. From being very close to the intended user experience, the development itself developed somewhat organically in both ways—from a backend logic perspective right through to front-end interaction design, with every technology choice supporting the greater goal of institution verification workflow simplification.

5.2 Feature Engineering

The Institution Ranking Checker feature engineering procedure was primarily guided by two competing objectives: firstly, to mimic the essential behaviour of manual document verification, and secondly, to increase amenity by applying automation wherever desirable. The designer psyche was not to bombard the users with an array of sophisticated features but to seek out the existing workflow's most critical pain points and attempt to alleviate those with a solid set of practical functionalities.

One of the most important features is the institution ranking management system, which will allow administrators to upload customised ranking datasets in Excel format (Loukides, 2011). The rank system has been developed keeping in mind operational flexibility, accepting that institutional benchmarks used during different admission cycles or by varying groups of users may vary slightly. Upon upload, ranking data is parsed and displayed transparently to the users, making available a listing of ranking sheets and some metadata associated with them (e.g., upload dates, dataset scope). More importantly, all ranking data is stored in memory only for the duration of the session, in alignment with GDPR and Trinity College Dublin’s data governance policies.

In contrast, the document upload module allows for uploading transcripts, CVs, and recommendation letters in various file formats: PDFs, DOCX documents, and scanned images in JPG or PNG. Since formats of admissions documents vary widely, supporting this

variance was considered a core feature, consisting of Textract support for normal documents and Tesseract OCR support for scanned images. Once uploaded, documents would be automatically processed to extract the names of relevant institutions and promptly matched against the ranking dataset. Essentially, an easy profiling experience was provided for admission staff, who often have to process a flurry of documents under time constraints.

To assist more in decision-making for admissions, the system includes an intuitive manual search function to search the ranking database directly with smart autocomplete suggestions that provide real-time hits based on partial input (Li et al., 2020). The manual search is a verifier and very helpful when there are ambiguous results or for reviewing documents when the automated matching algorithms yield uncertain results.

A history tracking function supports this by providing a live log in the session of all uploaded documents and processed results. Users can revisit past actions for comparison of outputs and cross-referencing of information within the same session, without compromising data privacy. This design encourages iterative checking and workflows where multiple documents are reviewed in quick succession.

An additional feature was the immune response to the one-click-shutdown mechanism, formally implemented for draining all temporary data safely and giving users assurance that no residual data traces remained from their interaction with the system. This functionality came forward as a direct response to stakeholder emphasis on data security and system transparency.

In brief, what the combined features offer are acceleration, simplicity, and clarity. Each feature was valued not just by its own stand-alone technical merits but also by what that particular feature and its technology meant to actually getting admitted teams through their work-a-day processes quickly with as little fuss as possible.

5.3 Backend Architecture and System Setup

The backend architecture of Institution Ranking Checker was strongly motivated by considerations of modularity of system components, security of sensitive data, and openness in terms of an audit trail (Bass et al., 2012). The architecture was deliberately designed to follow a logical workflow of the institution verification process from document ingestion to data extraction, matching, and finally, result presentation.

Fundamentally, the system architecture consists of clearly delineated functional modules, each of which fulfils a function associated with a particular stage in the workflow:

- Document Parsing Module (Textract/Tesseract)—Responsible for extracting text from a wide range of document types, thus ensuring compatibility with both digital and scanned formats.
- Institution Matching Module—Carries out the semantic matching by AI (through LLaMA) and semantic matching by fuzzy matching method so that the system will be robust against all sorts of variations in document contents and in the actual quality of the document samples.
- Ranking Dataset Management Module—Handles uploading, temporary storage, and querying of institutional ranking datasets using Pandas. It lets us achieve a very flexible ranking dataset handling yet session-bound data minimisation.
- Session Management & Data Handling Module—Handles tracking of user interactions, uploaded files, extracted data, and search queries within an active session. All temporary data is cleared upon shutdown to maintain commitments to data privacy.

This modular architecture accomplishes much:

- Easier maintenance and scalability
- Easier debugging, fault isolation
- Stronger security boundary enforcement between modules

Error handling was the other focus of backing design. The system contains exhaustive exception management routines, so if extraction or matching has an error, it will not bring down the application but will, instead, give meaningfully appropriate responses back to the user, thereby maintaining its resilience (Fowler, 2004).

Since this AI-driven workflow is designed to take precedence whenever it's possible, it will guarantee fallback if for some reason the AI module fails or returns uncertain results, so that there will always be at least some basic system reliability guaranteed in always being able to perform in offline or constrained environments.

All backend processes operate locally on the user's machine, providing further backing to the whole project in terms of keeping in line with GDPR and institutional data protection

regulations. Temporary files, results from processing by inference, and contextually sensitive data only exist during use and are all discarded in a secure manner once the session is over.

Being, at its core, an architecture on the backend whose very design aimed to deliver not purely in terms of technology but in harmony with the bigger picture approach embraced by this project—encompassing transparency, modularity, security, and user control—should therefore ensure the system delivers, not just in operational reliability, but also in ethical data handling practices.

5.4 Frontend Interface Design

The description of the design of the frontend interface of the Institution Ranking Checker brings forth the concept of a truly user-centred Ramachandran: to provide an ideal experience for an admissions office that might feature few technical staff among them. The interface hinders the user-friendly interaction with the backend of the system while creating conditions conducive to transparency, control, and an easy flow of navigation.

The interface includes a fixed top navigation bar that unites all the modules of the system and hence ensures user attention on the process at hand without interference. The critical system functions—Home, Manage, Upload, Results, Crosscheck, Search, History, and Shutdown—are easily accessible via the modules. Users can jump from one module to another without having to retrace all their steps, and this usability feature greatly improves user efficiency. The TCD logo is set prominently in the header to reinforce institutional branding and augment credibility to the tool's interface.

The Home Page doubles as a dashboard and a guide for beginners. It explains the system’s goal, its main features, and the documents required. It describes "How It Works" by going through the entire workflow in a straightforward way so that users are ready for everything that will happen from there on. The home page lists all inputs and outputs and helps to prevent user errors while setting expectations for what the system can do.

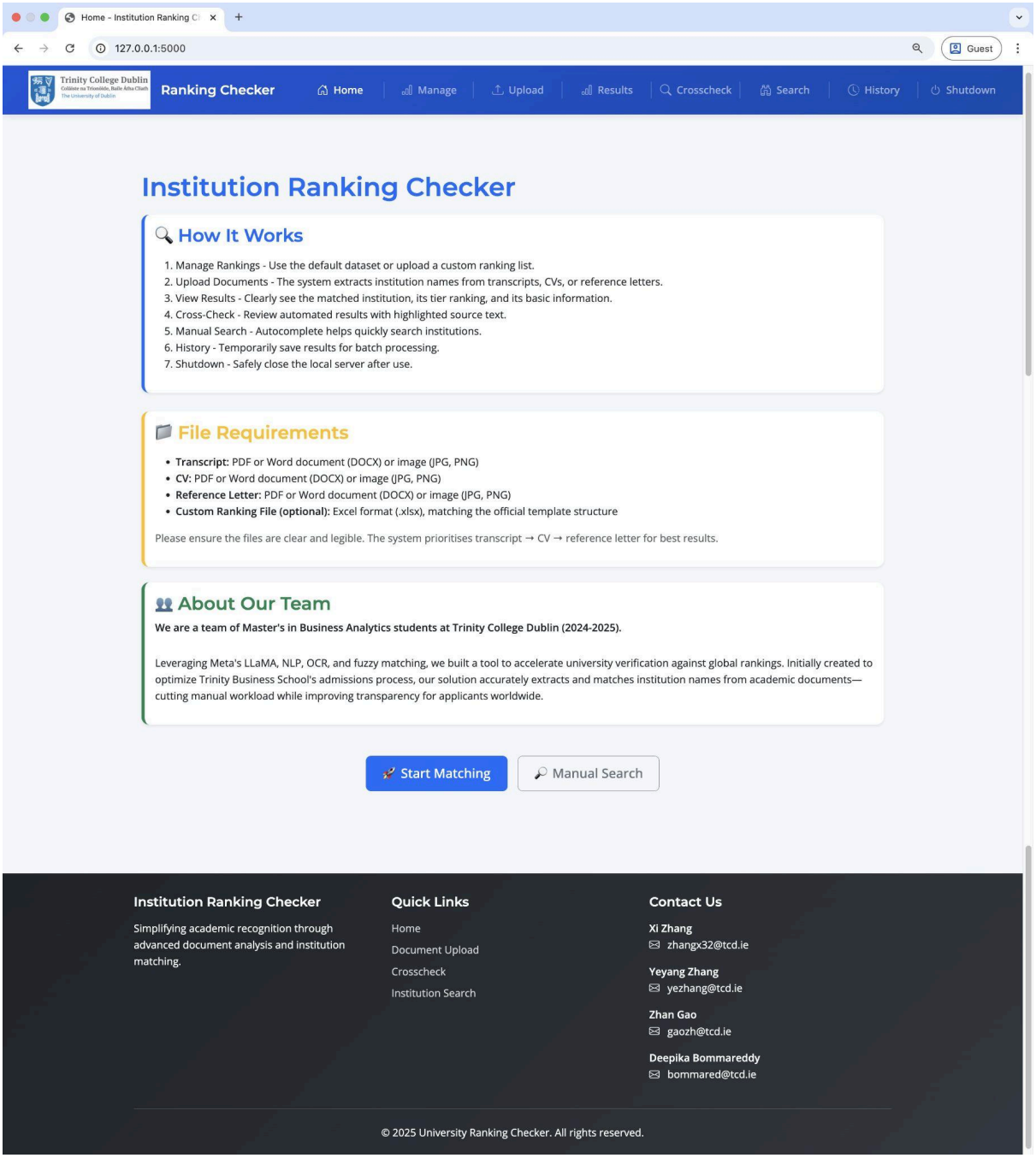

Figure 5: Interface - Home Page

On the Manage Page, ranking datasets are uploaded and managed. The page provides an interface for adding ranking files and displaying the existing files with necessary details such as name, sheet information, and date and time of uploading. The UI is deliberately uncluttered so the users can concentrate on ranking system selection and management. This further underlines a commitment to flexibility and user control in terms of being able to view and manage ranking systems in one location.

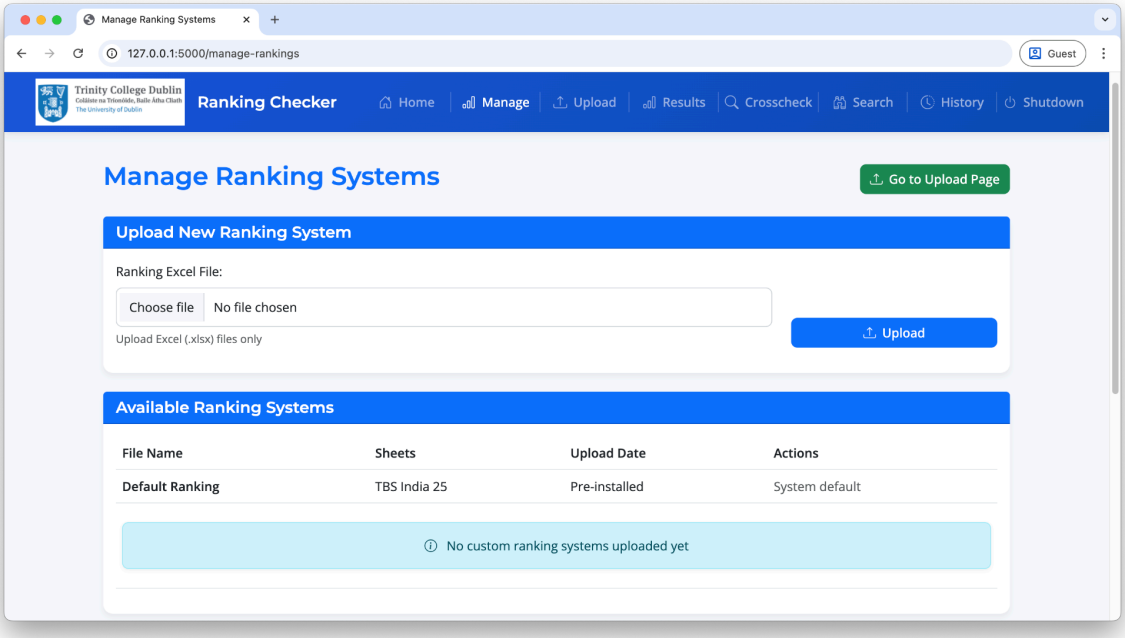

Figure 6: Interface - Manage Page

The Upload Page makes up its mind about submitting documents. In other words, since the view for uploading transcripts, CVs, and reference letters are altogether combined, it mirrors the daily life workflow of admissions staff in processing materials for applications. The Web page also provides dropdown boxes that allow users to select the currently active ranking system (with which each uploaded document will be associated). The processing starts on a click of a centralised Submit button, and a friendly error message will direct the user when an input format is wrong.

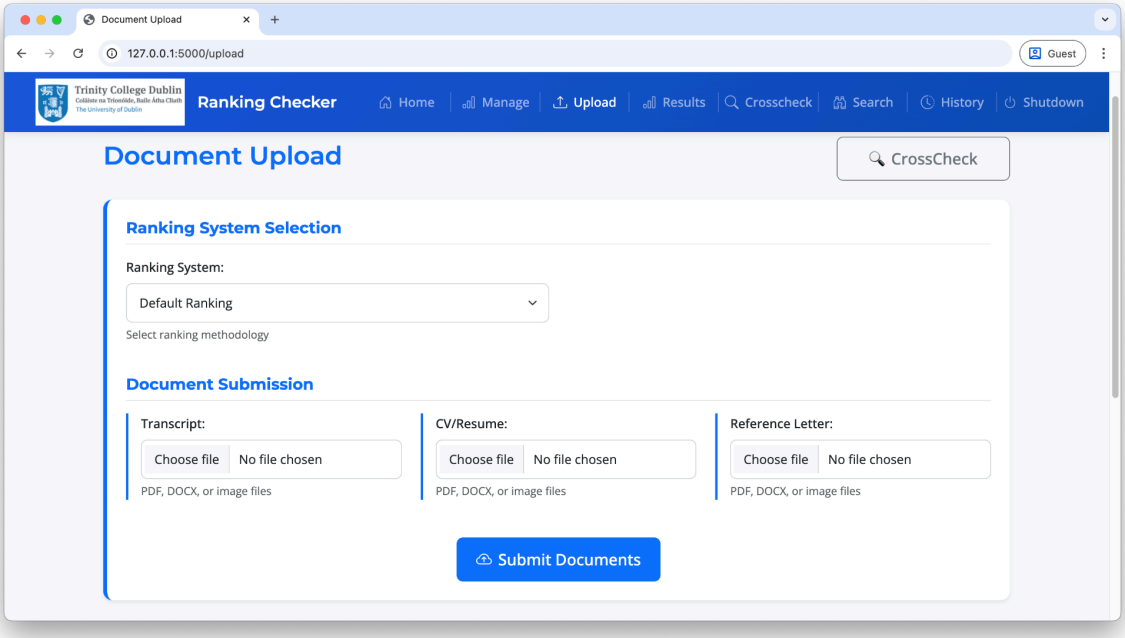

Figure 7: Interface - Upload Page

The Results Page meant output display very clearly and provided transparency into result processing. As it stands, it displays extracted institution names from uploaded documents, with the evidence indicating whether the match was through the LLM-based method or the traditional rule-based method. Some fields with scarce evidence are equipped with an expandable toggle option to see even more contextual clues to build transparency in the minds of sceptical or curious reviewers. The design of this evidence hierarchy allows it to be scanned intuitively—both for quick reviews and for a deeper dive.

Based on the document upload and extraction event, the system would dynamically change. Should the document be valiantly uploaded and the LLM or the traditional method extracts some valid institution-related information, the Results Page shall furnish them with a detailed summary containing the institution name and various metadata regarding such as city, state, and degree level; who extraction method was used (LLM or the traditional method); the institution ranked tier classification; and also textual evidence supporting the matching. Such an unstructured output enables academic staff to reliably verify and apply the extraction information more efficiently.

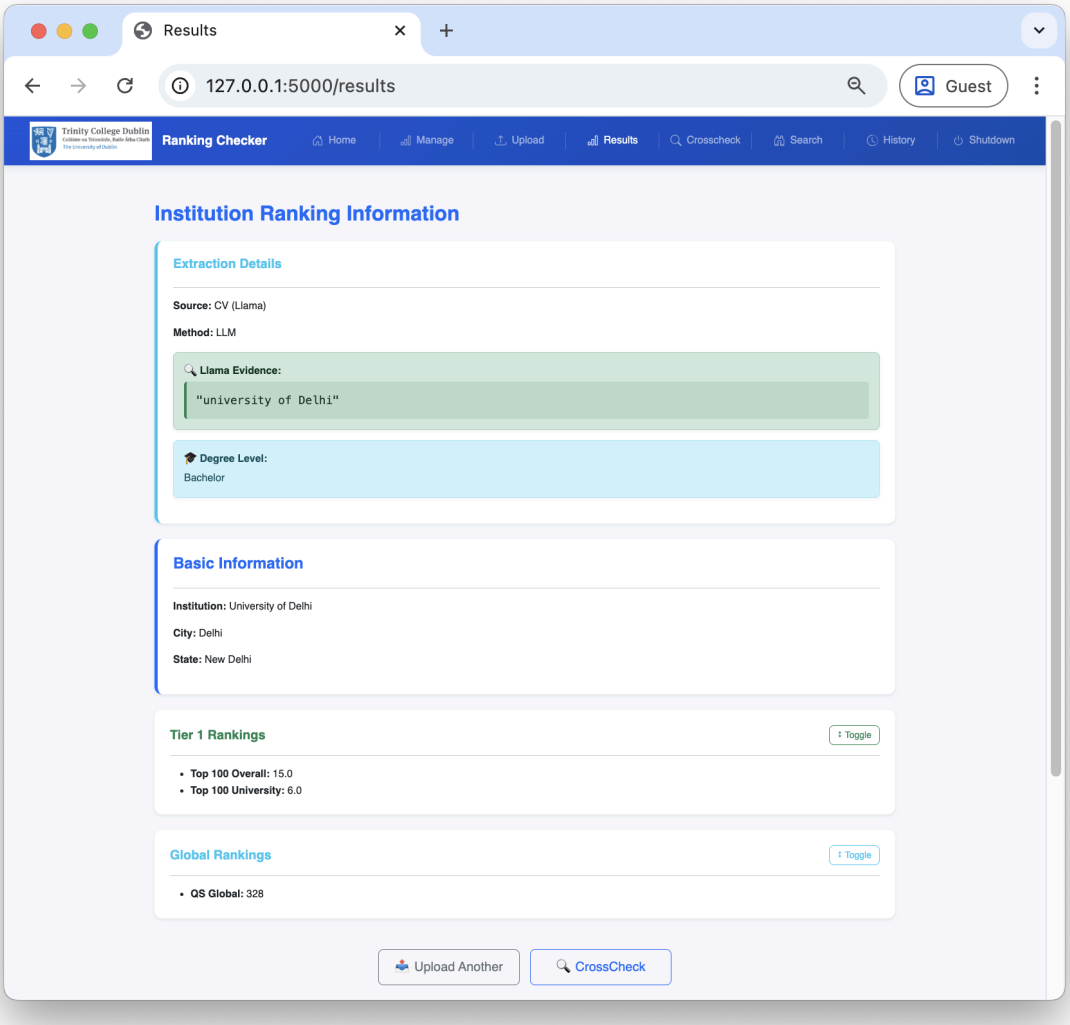

Figure 8: Interface - Result Page (if match)

In case the upload is not successful, or both extraction methods from data do not validate enough institution information, an indication that "No institution was matched" is shown on the Results page; users are then advised to double-check the uploaded document or consult the CrossCheck page for further assistance. Essentially, this concept helps ensure that users are made aware of the extraction procedure's limitations yet allowing session continuity for further exploration or remediation.

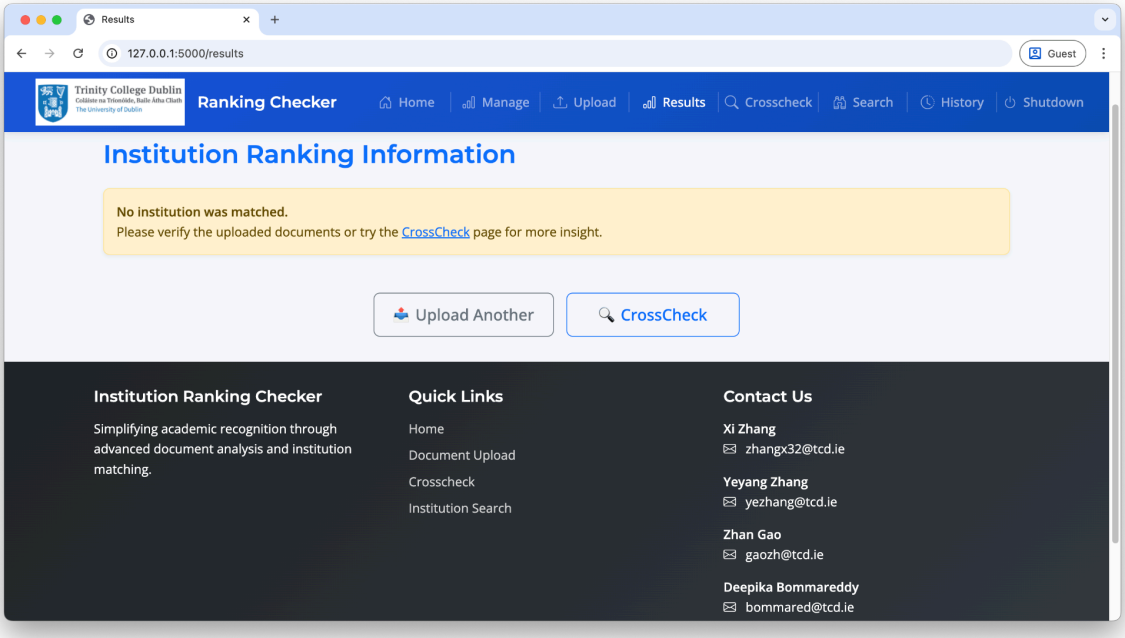

Figure 9: Interface - Result Page (If no match)

The Manual Crosscheck Page facilitates comprehensive verification. Users can review the content of matches' texts and extracted documents in detail. This promotes accountability and allows users to challenge or affirm the results produced by the automated system. The page layout shelters maximum ease of readability and time to critical information.

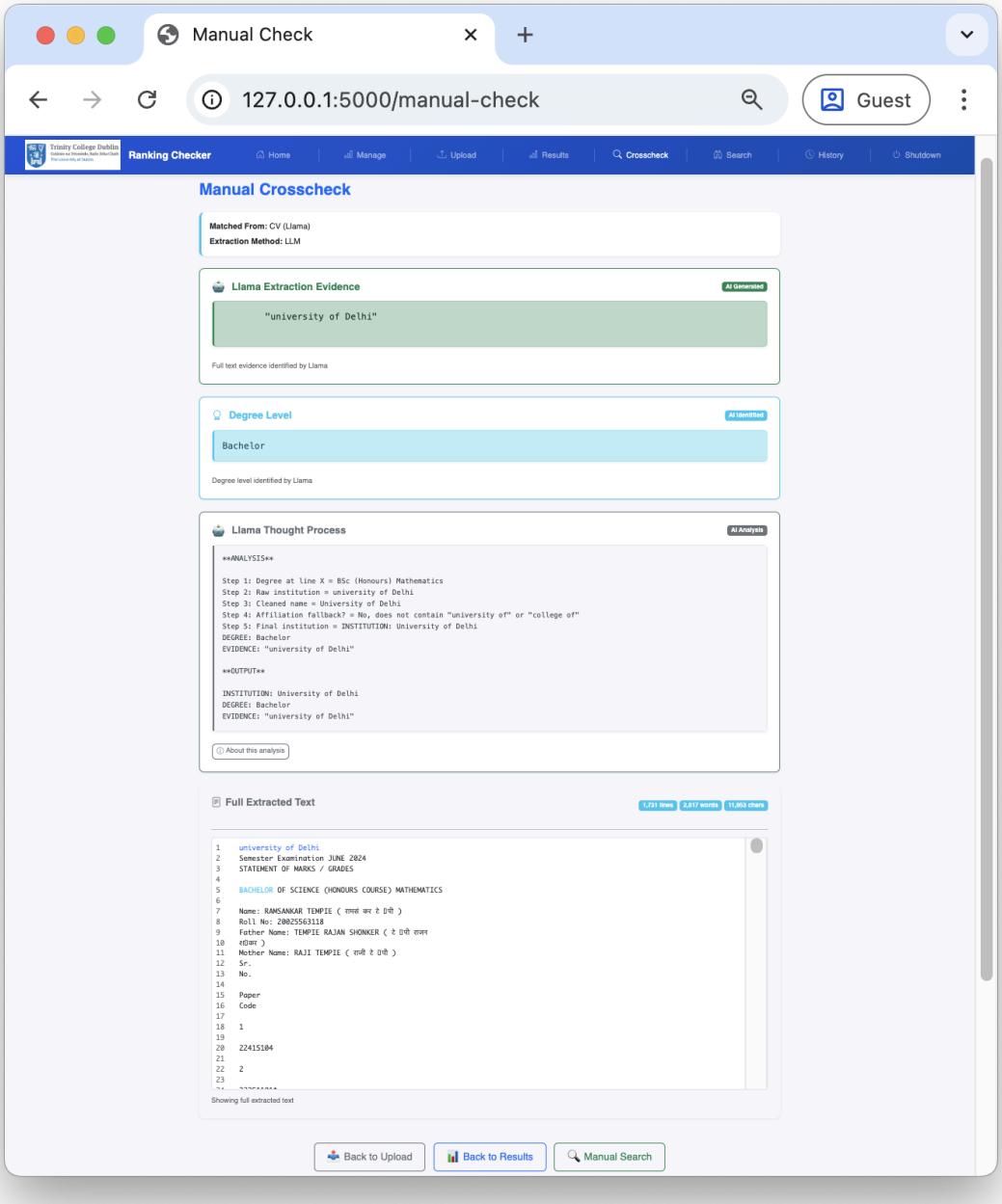

Figure 10: Interface - Crosscheck Page

The Search Institution Page provides the ability to manually search within the ranking dataset. The input supports partial matching and displays dynamic suggestions via an autocomplete function. This gives users the ability to quickly search for institutions rather than scrolling through the data set or going to external sources, thus facilitating the process of verification and data lookup.

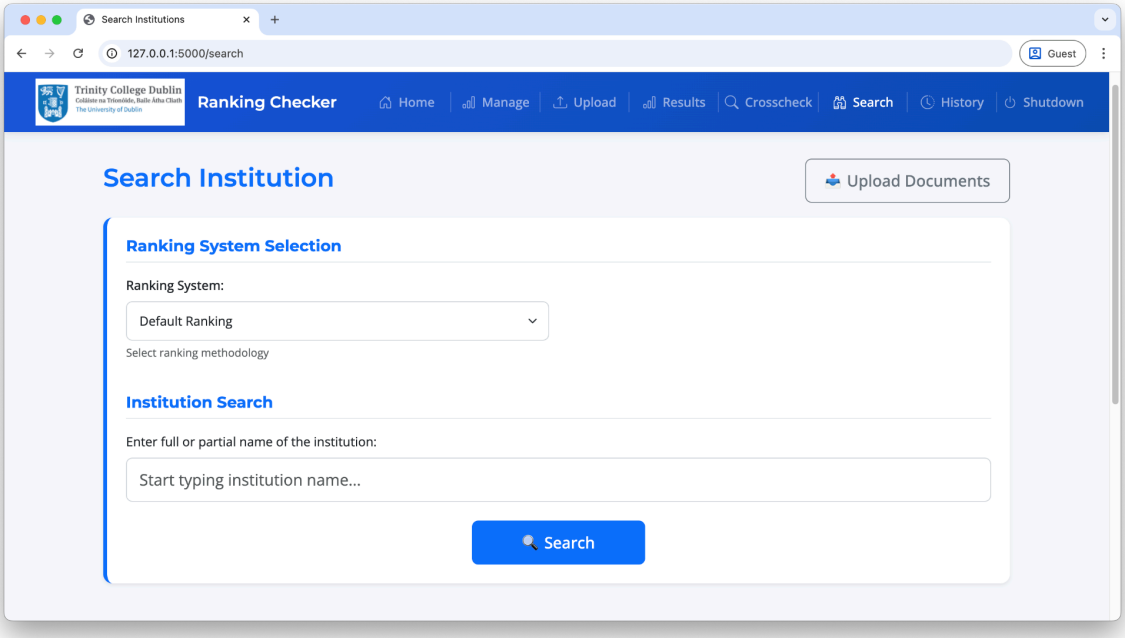

Figure 11: Interface - Search Page

The History Page provides an in-session record of all document uploads and associated processing activities. This feature allows admissions personnel to easily track and revisit their document verification attempts within the current session, which is particularly beneficial for batch processing or reviewing multiple applicants. Each upload entry displays timestamped file names, the ranking system used, and the match result.

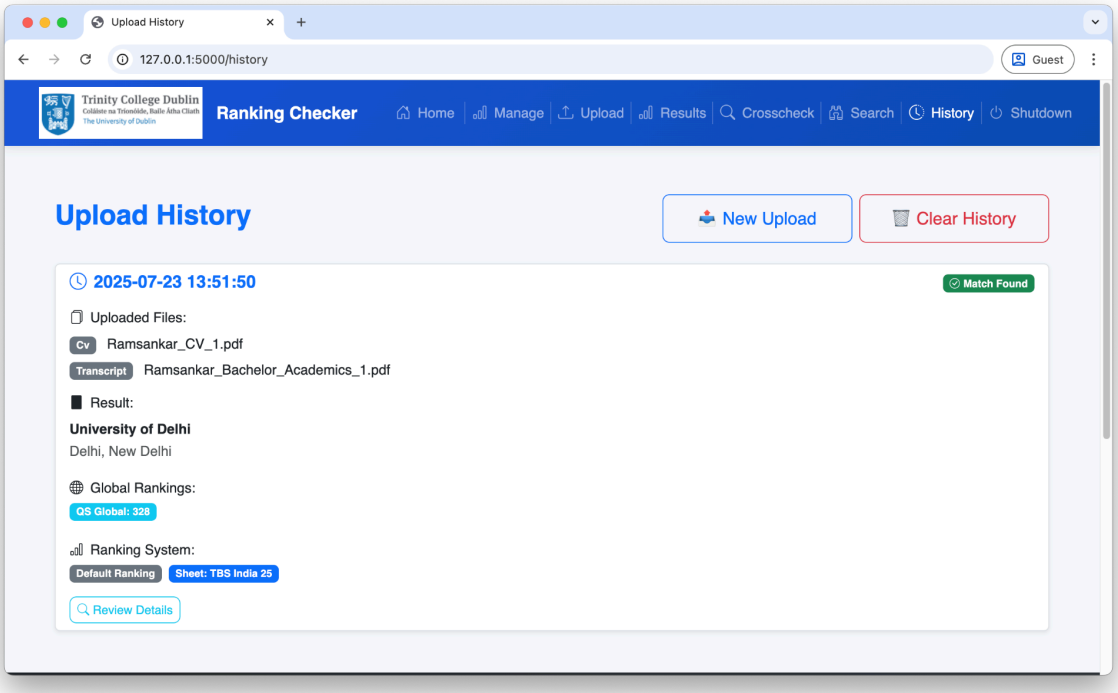

Figure 12: Interface - History Page

To support data minimisation principles, no information persists beyond the current session. Users have the option to clear the session history using the Clear History button, which resets the interface and displays a confirmation message and an empty state. This design ensures transparency in document tracking while maintaining strict session-based privacy protocols.

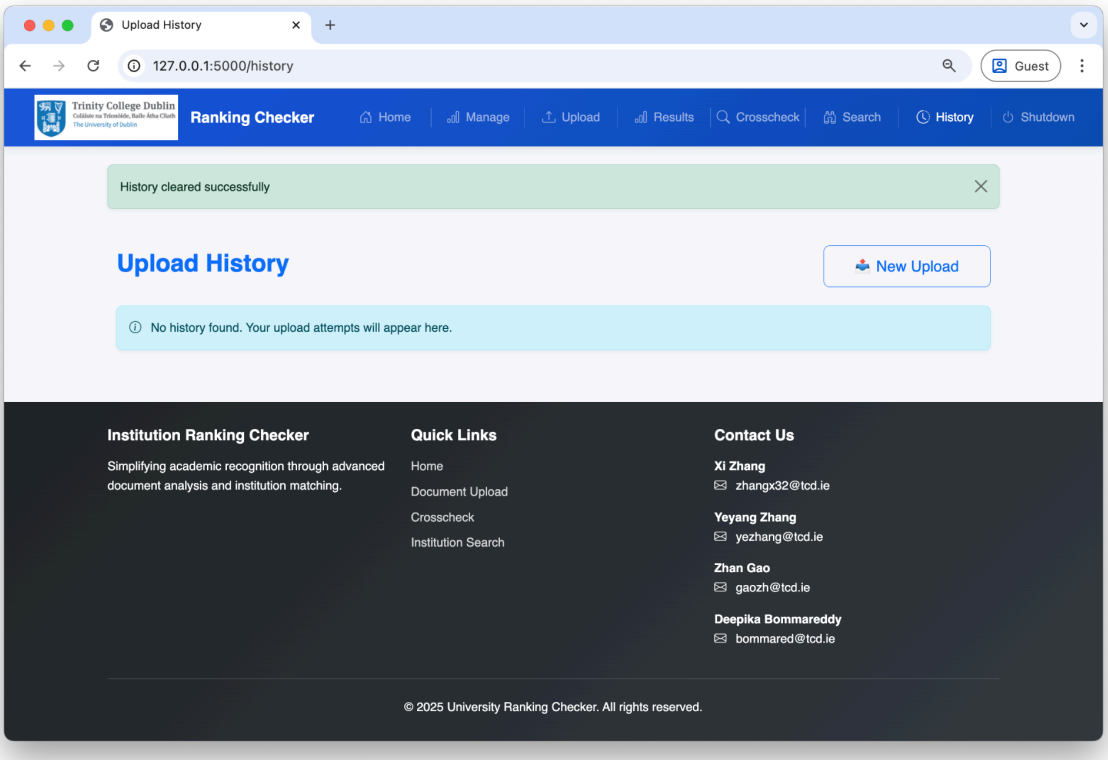

Figure 13: Interface - History Page (clear history)

The Shutdown Option offers a simple, visible mechanism for terminating the application. Activating it will clear all session data, stop the server, and cleanly exit the application. This is indicative of how much emphasis the system places on user control, data security, and ethics in software development.

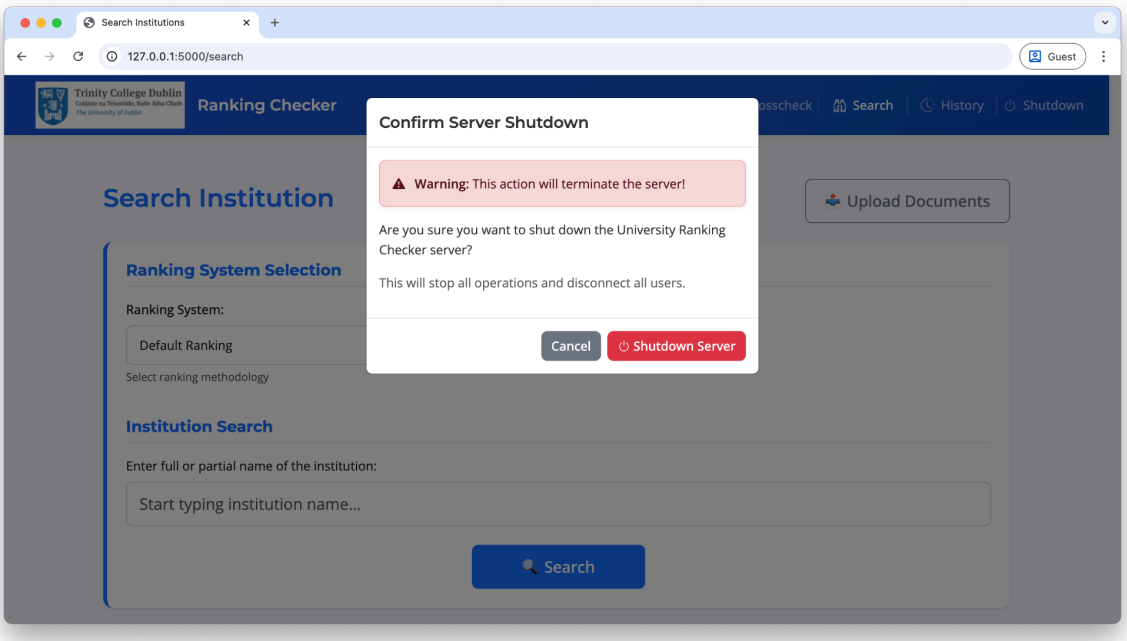

Figure 14: Interface - Shutdown 1

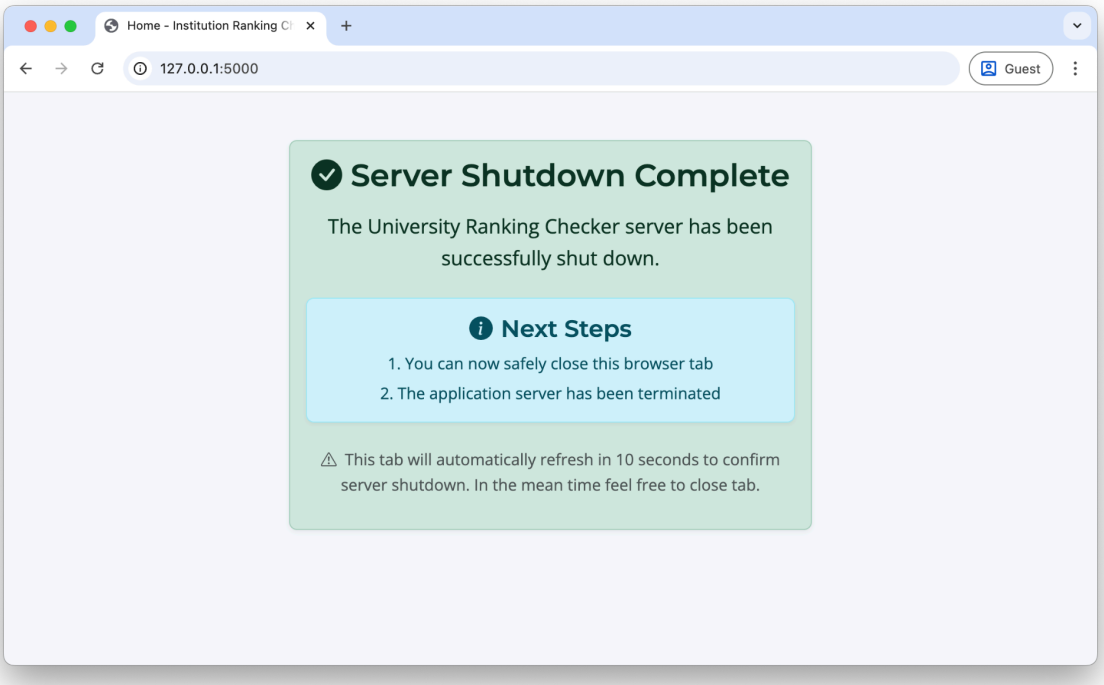

Figure 15: Interface - Shutdown 2

Thus, the frontend interface was designed not merely as a functional layer but as a thoughtful extension of the system's operational philosophy—combining accessibility, transparency, and control within a coherent, user-friendly environment. Every design decision, from navigation flow to information display, was made with the practical needs of admissions staff in mind, ensuring that the system supports their work effectively without introducing unnecessary complexity.

Chapter 6: Testing and Iteration

6.1 Functional Testing

The functionality of the Institution Ranking Checker was tested to assess whether the system's core components operated as intended in the most common setting reflected from the point of view of the Trinity Business School admission officers. The team took a pragmatic approach to testing, rather than applying a rigorous software testing methodology –

reflecting the true end user's workflow to processing admissions. Attention was given to proving the application’s core functionality: uploading and parsing applicant documents, extracting institution names with either fuzzy logic or an LLaMA, matching the institution names against a structured ranking database and serving up the results in a clear, user-verifiable format. Real university admissions documents were tested, such as digitally constructed transcripts, CVs, recommendation letters, and scanned images. The cross platform reliability of each module was verified on macOS and Windows, and all processing occurred on the local machine, thus retaining the GDPR compliant status of the system.

The most crucial features tested were the document upload system that accepted PDFs and other image formats, like JPG or PNG. The document parsing pipeline (served with Textract for digital and Tesseract for image) extracted text and was ready for processing. Also, the institutional ranking upload module was tested by means of Excel files as described in chapter 7 with the predefined column layout. Alongside automatic detection capabilities, a manual search and verification system function was tried to check the usability and precision. We also verified logging of session history, and the secure shutdown function to clear temporary data from the system after a session was terminated, thus maintaining the systems' privacy-by-design aspect. Overall, individual components were tested separately and within the whole workflow to check if the correct integration and execution is realised and serves as the basis for a detailed results analysis.

6.2 Test Cases and Outcomes

The Institution Ranking Checker was evaluated on a dataset of 42 structured real-world admissions documents and two institution name extraction methods - fuzzy string matching with the RapidFuzz library and a large language model (LLaMA) based on the Ollama framework were used. The test dataset consisted of common admission documents such as CVs, transcripts, recommendation letters, degree certificates. The purpose of this evaluation was not only to compare the raw accuracy of each system but also to ascertain system performance under production conditions that replicate the real-life application used for the Trinity Business School admissions process.

The system is purposefully constructed to measure Indian institutions in the context of higher education in India. With cases such as the University of Hertfordshire (UK University), and University of British Columbia (Canadian University), the system provided the absence of a

match as expected. This behaviour was consistent across both models and reflects the intended scope of the tool, which is to assist with the evaluation of applicants from Indian universities only.

The fuzzy matching procedure achieved correct or acceptable results for 19 of the 42 test cases, with an overall accuracy rate as 45%. It performed well on well-structured documents like transcripts or plainly formatted CVs. For example, test ID 19326759 in which “Amity University” was well extracted by virtue of clean layout and explicit presence of institution. Likewise, when the affiliated institutions were referred to as Veermata Jijabai Technological Institute (VJTI), the system predicted the parent institute as University of Mumbai (ID 25333210). But the application of fuzzy matching started to reach its limit when it comes to complex or unstructured document types. Seven documents were completely missed due to either OCR problems (poor input layout) or unclear layout, e.g., in IDs 25332126 and 25355469, the input may have been low quality and the extraction failed. In 14 additional cases, fuzzy matching returned wrong results due to its over-dependence on token similarity where high-frequency words like “college,” “engineering,” or “university” biased the matching logic. For example, in Ref. ID 25359439, the obtained answer was “University of Science and Technology” instead of “N.M.A.N. Institute of Technology”, whereas for Ref. ID 25341865, it gave “University of Delhi” instead of NMIMS because of the overpresence of the word “University" in the text. The fuzzy matching did not perform well at discriminating between unrelated but similarly named institutions, nor schools when the name was found.

We found that traditional fuzzy matching techniques can not accurately identify institution names within the extracted text data containing significant noise. So we decided to incorporate LLaMa, a Large Language Model to give better results.

In comparison, the LLaMA-based language model had a more powerful and context-dependent extraction capacity. Of those same 42 test cases, LLaMA kernel-mediated matching was correct or acceptable in 30 cases, with an overall accuracy as 71%. It has been performing particularly well on free-text documents or those with no particular formatting. For example, correctly identified “Indian Institute of Technology Madras” in ID 25359189 from a CV where a fuzzy match was unsuccessful. In other cases, such as IDs 25333210 and 25333924, LLaMA inferred the parent institution “University of Mumbai” from constituent colleges, demonstrating an understanding of institutional hierarchies that was absent in the string-based approach. The model also worked well in a few of the edge cases, for example

ID 25351025, where it correctly extracted “O.P. Jindal Global University” just from the CV. Importantly, the model returned “no match” in 12 cases—often when the institution was either not present in the ranking dataset or when the extracted name, while semantically accurate, had no corresponding entry. For example, in ID 25359387, LLaMA extracted “APJ Abdul Kalam Technological University, Kerala,” which was correct in context but not included in the database, and therefore appropriately returned as unmatched. This shows the model’s transparency and reliability in avoiding false confidence.

Comparison between the two methods (overall accuracy rate of 71% for LLaMA and 45% for traditional fuzzy matching, representing a 16% point improvement) demonstrated explicitly why LLaMA was developed and selected as the optimised detection approach. While fuzzy matching worked well enough as a baseline on clean, well-formatted documents, it was not semantically rich enough to handle the variation seen in actual admissions documents. Language understanding in LLaMA allowed for more accurate resolution of institutions embedded in narrative text, more graceful handling of OCR noise, and to infer parent institutions where possible. These capacities made it more capable of handling the complex, high-stakes job of reviewing admissions. Furthermore, the system design includes a fallback to fuzzy matching in cases where LLaMA fails, providing an added layer of robustness. Throughout the testing, the system remained stable with no crashes or unexpected behaviour. All operations were carried out locally, ensuring offline functionality and maintaining GDPR compliance. Sessions were logged, and all temporary data was cleared upon termination, aligning with the system’s privacy-by-design principle.

Finally, rationale methods Nevertheless both approaches led to the development and improvement of the institution extraction pipeline, however, the better performance and ability for reasoning about the context of LLaMA made it the method of choice for production use. Its improved matching rate, lower false-positive rate, and robustness against real-world document variability greatly improved the usability and reliability of the tool for TBS’s admissions process.

Chapter 7: Deployment and Implementation

7.1 Excel File Format Specification

To ensure robust data integrity, seamless backend compatibility, and accurate automated ingestion, all Excel files integrated into the system must strictly conform to a predefined structural format. Any inconsistency or deviation from the specified schema may result in data processing failures, incorrect parsing, or overall system malfunction during import operations.

7.1.1 Structural Overview

The Excel spreadsheet is organised around two hierarchical ranking tiers:

- **Tier 1:** Represents institutions ranked within the global top 100 across multiple academic and institutional categories.
- **Tier 2:** Encompasses institutions ranked from 101 to 200 within equivalent categories.

Each tier contains a set of distinct ranking criteria, mapped to dedicated columns. This columnar segregation enables precise identification, categorisation, and interpretation of each institution's performance during automated backend ingestion.

7.1.2 Required Column Headers

To ensure compatibility with the application's parsing logic, the spreadsheet must include the following column headers in the exact order listed below. These headers are case-sensitive and must not be altered, renamed, removed, or rearranged:

- Name of Institution
- CITY
- STATE
- Tier 1
    - Top 100 Overall
    - Top 100 University
    - Top 100 College

    - Top 100 Engineering
    - QS World
- Tier 2
    - 101–200 Total
    - 101–200 University
    - 101–200 College

A	B	C	D	E	F	G	H	I	J	K
			TIER 1					TIER 2		
Name of Institution	CITY	STATE	Top 100 Overall	Top 100 University	Top 100 College	Top 100 Engineering	QS Global	101-200 Overall	101-200 University	101-200 College

Figure 16: Sample Excel Format Layout

Each row in the spreadsheet represents a unique educational institution, along with its corresponding attributes across these ranking categories. The format adheres to a strict column order and naming convention to facilitate seamless data parsing and integration within the application's backend.

7.1.3 Formatting Guidelines

In order to guarantee reliable ingestion and maintain system stability, the following formatting rules must be observed:

- The header row must be placed as the first row of the worksheet and must match the standard header structure exactly.
  All institutional data must begin on the second row, directly beneath the header, and be fully aligned with the corresponding columns.
- No additional columns (including hidden or auxiliary metadata) are permitted within the file.
- Each row must correspond to a single, unique academic institution.
- Data values must maintain consistent formatting; for example, ranking fields should not mix numeric and textual data (e.g., avoid mixing "95" and "Ninety-Five").
- Merged cells, formulas, or special formatting should be avoided to prevent parsing inconsistencies.

This strict format adherence guarantees that the data import module properly reads and processes the Excel file automatically. Non-compliance with the standard format might cause processing errors or insufficient ingestion of data.

7.2 System Installation and Configuration Guide

This guide provides instructions to configure and launch a Flask-based application integrated with the LLaMA (via Ollama) language model for intelligent document processing, including PDF parsing, OCR, and AI-powered responses.

7.2.1 System Components and Dependencies

The following tools and libraries are essential for full functionality:

Component	Purpose
Python 3.9+	Core programming environment
Flask	Web framework for application backend
Ollama + LLaMA	Local LLM engine for AI inference
Poppler	PDF to image conversion
Tesseract OCR	Image to text extraction
Required libraries	flask, pandas, pytesseract, textract, six, pdf2image, nltk, ollama, openpyxl

7.2.2 Deployment Instructions for macOS

This section details how to install and configure the application on macOS.

Step 1: Install Python (Version 3.9)

- Download Python 3.9 from the official site: https://www.python.org
- Run the installer and follow the on-screen instructions.
- After installation, verify with:

```
None
python --version
```

Expected output: Python 3.9.x

Step 2: Install Ollama (LLaMA Interface)

- Visit: https://ollama.com
- Click Download for macOS and install the application.
- Once installed, run Ollama from Applications to ensure it's set up.
- In Terminal, verify by running:

```
None
ollama run llama3
```

This command will automatically download and initialize the LLaMA model if it hasn't been pulled yet.

Step 3: Install Homebrew (if not installed)

Homebrew is required for installing packages like Poppler and Tesseract:

- To install Homebrew, open Terminal and run:

```
None
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

- After installation, confirm with:

```
None
brew --version
```

Step 4: Install Poppler (for PDF processing)

Used to convert PDFs into images for OCR:

- Install it using Homebrew:

```
None
brew install poppler
```

This makes the pdftoppm and pdftocairo tools available to users' system.

Step 5: Install Tesseract OCR

Tesseract is used for Optical Character Recognition (OCR) of scanned or image-based documents.

- Install it via Homebrew:

```
None
brew install tesseract
```

- To verify installation:

```
None
tesseract --version
```

Step 6: Install Required Python Libraries

- Use pip to install all necessary Python packages. In Terminal, run:

```
None
pip3 install flask pandas pytesseract textract six
pdf2image nltk ollama openpyxl
```

This will install the backend dependencies for document parsing, OCR, ranking lookup, and web service management.

Step 7: Download the LLaMA Model Using Ollama

- If not done earlier, pull the model manually:

```
None
ollama pull llama3
```

Once downloaded, the model is available locally and does not require an internet connection for further use.

Step 8: Running the Application

- Navigate to users' application directory:

```
None
cd /path/to/your/project
```

- Then start the Flask server:

```
None
python3 app.py
```

Access the application at http://localhost:5000.

7.2.3 Deployment Instructions for Windows

To run the Flask-based application with LLaMA integration on a Windows system, follow the steps below to configure the application environment correctly:

Step 1: Install Python

- Visit the official website https://www.python.org and download Python 3.11.
- During installation:
    - Check the box labeled "Add Python to PATH"
    - Proceed with installation using default settings.
- After installation, confirm by opening Command Prompt and typing:

```
None
python --version
```

Step 2: Install Ollama (LLaMA)

- Download and install Ollama from: https://ollama.com
- Make note of the installation directory (e.g., C:\Users\YourName\AppData\Local\Programs\Ollama)
- After installation, open a new Command Prompt and run:

```
None
ollama run llama3
```

This will download and run the model locally.

Step 3: Install Poppler (for PDF to image conversion)

- Download Poppler for Windows from: https://github.com/oschwartz10612/poppler-windows/releases
- Extract the contents and retain the full path to the bin directory (e.g., C:\Tools\poppler-xx\bin).

Step 4: Install Tesseract OCR

- Download Tesseract from: https://github.com/UB-Mannheim/tesseract/wiki
- Install the software and remember the installation path.

Step 5: Configure Environment Variables

To ensure command-line access to all required tools:

- Open System Environment Variables:
  - Right-click on This PC → Properties → Advanced System Settings → Environment Variables
- Under System variables, find and select the Path variable, then click Edit.
- Click New, and add the paths to the following if not already present:
  - Python (e.g., C:\Python311\, It should be done if "Add to path" is selected while downloading Python.
  - Ollama (e.g., C:\Users\YourName\AppData\Local\Programs\Ollama)
  - Poppler's bin folder (e.g., C:\Tools\poppler-xx\bin)
  - Tesseract.exe folder (e.g., C:\Program Files\Tesseract-OCR)
- Click OK to apply the changes.

Step 6: Install Required Python Libraries

- Open Command Prompt (cmd) and run:

```
None
pip install flask pandas pytesseract textract six
pdf2image nltk ollama openpyxl
```

Step 7: Download LLaMA Model Using Ollama

- Run the following command to download and prepare the LLaMA model locally:

```
None
ollama pull llama3
```

This will download and prepare the model.

Step 8: Run the application

Open Command Prompt (cmd) and run:

- Navigate to your application directory:

```
None
cd /path/to/your/project
```

Users can get the project file path by pressing “Ctrl+Shift+C”.

- Then start the Flask server:

```
None
python app.py
```

Access the application at http://localhost:5000.

7.2.4 Snapshot: Configured System

The following figure 17 shows the expected appearance of the terminal when the system and environment have been correctly configured. Although this screenshot was captured on a

macOS device, the interface and terminal output are consistent across both macOS and Windows platforms, as verified through cross-platform testing.

Figure 17: Terminal Output from a Successfully Configured System

Figure 18 displays the system behaviour when the Llama model is unavailable or encounters an error. The terminal clearly indicates a "Llama error", allowing users to understand the cause. Despite this, the application remains fully functional by automatically falling back to the fuzzy matching method for name extraction. This ensures uninterrupted operation, even in environments where the LLama model cannot be executed.

Figure 18: Terminal Output When LLama Model Is Unavailable

7.2.5 Common Issues and Troubleshooting

This section outlines common issues encountered during system setup or execution, along with recommended solutions.

- **File Not Found/ModuleNotFoundError**

Ensure that all required tools and Python packages are correctly installed, and that their paths are properly configured in the system's environment variables.

- **Failed Text Extraction**

Verify that both Poppler and Tesseract are installed and accessible via the command line. These components are essential for accurate PDF and image text extraction.

- **Model Errors or Incomplete Responses**

Confirm that the LLaMA model has been fully downloaded using Ollama, and that the host system has sufficient memory and computational resources to support inference.

Additional Notes:

- If the pdftoppm or tesseract commands are not recognized, ensure that the /opt/homebrew/bin/ directory is included in the system's PATH variable (particularly on macOS systems).
- For Python-related errors, consider reinstalling the affected packages using the --upgrade flag to ensure compatibility and the latest version:

```
None
pip3 install --upgrade package_name
```

- If changes to environment variables do not take effect immediately, restarting the terminal session or the entire system may be necessary.

Chapter 8: Limitations

The current version of the application demonstrates a robust approach to institution recognition and document-based information extraction. However, there are several limitations that affect the overall performance, accuracy, and operational efficiency of the system. These limitations are both technical and practical, and recognizing them is essential for guiding future iterations of the system. This chapter outlines key areas where the application currently falls short and provides context for potential improvements.

8.1 Incomplete Institutional Coverage

The app uses a fixed, manually maintained Excel based ranking list that excludes a lot of legitimate institutions, especially independent colleges, upstart universities, and international

institutions. Therefore, about 10% of the test data returned was not found even when the model identified the correct institution. This issue can significantly affect results in real-world use cases where applicants come from a broad range of educational backgrounds, especially those Indian students from non-traditional or foreign institutions.

8.2 OCR Sensitivity in Scanned Documents

Another major bottleneck is the system's dependency on Optical Character Recognition (OCR) to convert especially scanned or image-based documents into machine-readable text. The OCR engine currently in use shows considerable sensitivity to input quality. In cases where the documents are poorly scanned, low in clarity, or formatted in non-standard layouts and art font(e.g., tables, sidebars, headers), the OCR output often becomes invalid, corrupted or just fail to extract anything. Watermarked documents, a common format in official transcripts and certificates, or handwritten words, seen in Recommendation Letters, further exacerbate this issue by interfering with text clarity.

In several cases, the OCR produced either no usable text or blank information, rendering it impossible for the language model or fuzzy matcher to perform any meaningful recognition. This limitation is particularly impactful in document-heavy workflows, such as university admissions or credential evaluations, where the accuracy of extracted content is critical.

8.3 Ambiguity in Affiliated Institutions

Institutional affiliations introduce another layer of complexity that the current system does not handle effectively. In many cases, individual colleges function under a larger university. For example, colleges like VJTI (Veermata Jijabai Technological Institute) or VESIT (Vivekanand Education Society's Institute of Technology) operate under the University of Mumbai. However, they often maintain distinct identities and reputations in the academic ranking sheet, particularly in the context of admissions and rankings.

The system occasionally maps these colleges at the university level. This simplification leads to partial matches which may be accurate in a technical sense but insufficient in practice. For example, if a document mentions "VJTI" and “University of Mumbai”, the system may recognise it as "University of Mumbai," thereby omitting the college-specific identity. This could result in confusion or misranking in scenarios where the distinction between a college

and its parent university carries significance, or it may not be sufficient in cases where college-level rankings are required for admissions decisions

8.4 Delays and Hardware Demands in LLM Processing

The incorporation of LLaMA, a large-scale language model, has markedly improved the accuracy of institution recognition compared to traditional fuzzy matching. However, this improvement comes at the cost of increased computational overhead. Running inference locally, especially for high-volume or batch document processing, introduces latency of up to tens of seconds per document.

These performance constraints pose practical challenges. For instance, in a real-world admissions office scenario, staff may be required to process hundreds or thousands of documents in a single session. The latency and hardware requirements could significantly hinder operational efficiency and user experience in such cases.

8.5 Absence of Student Name in Output Results

Another limitation is that the current system does not include the student's name in the output results. While the primary focus of the tool is institutional recognition and validation, omitting student identification from the result set can hinder quick cross-verification in operational workflows. The absence of a clear student name may require additional manual effort to refer back to the original documents for confirmation.

This limitation, though not directly affecting the model's accuracy in institution recognition, impacts the overall efficiency and usability of the system in real-world administrative settings.

Chapter 9: Future Work

While the current version of the Institution Ranking Checker delivers its core value—enabling automated, privacy-compliant institution identification and ranking from Indian undergraduate documents—there are several directions for future improvement that could enhance its flexibility, scalability, and long-term utility. These future enhancements

focus not only on feature expansion but also on deepening the system's technical sophistication, all while preserving its local-first, offline architecture in compliance with GDPR and Trinity College Dublin's data governance standards.

9.1 Dynamic Ranking Data Updates

As institutional reputations evolve over time, the ability to maintain an up-to-date ranking database becomes critical. The current system relies on a manually uploaded static Excel sheet, which poses limitations in keeping rankings aligned with the latest evaluation standards.

- Proposed Feature: Introduce a dynamic update mechanism that allows administrators to seamlessly upload new ranking files while being prompted periodically when the current version is outdated.
- Technical Implementation: Monitor file age using filesystem metadata (e.g., last modified timestamp via os module). Add a UI prompt to suggest updates after a defined interval. Validate the structure of newly uploaded files using pandas and custom schema checks. Store a cache of older versions with timestamped filenames for rollback if needed. Optionally support downloads from trusted external sources for manual updates (e.g., QS, NIRF).

9.2 Multi-Country and Multilingual Document Support

The Trinity Business School is also receiving applicants from many other countries apart from India. Each country holds a set of institutions, languages, and document-formatting practices. To accommodate this plurality, the system must evolve to support global datasets and multilanguage issues.

- Proposed Feature: Allow multiple-country document processing wherein each country's ranked dataset and multilingual text extraction pipeline would be supported, positioning the system as a versatile international admissions tool.
- Technical Implementation: Create a modular ranking loader that allows users to select datasets by country/region. Store ranking data in organised directories (e.g., rankings/india/, rankings/china/). Use OCR libraries like easyocr, tesserocr, or pytesseract with multilingual support. Integrate multilingual NER or translation pipelines for non-English documents. Normalise naming conventions across countries

(e.g., “University of X” vs. “X Institute”). This direction positions the system as a versatile international admissions tool.

9.3 Support for Additional Document Types and Layouts

There's a lot more that real-world certain admission documents can entail other than transcripts and CVs. Scanned certificates, diplomas, awards, or language tests could also be submitted-is one way-typically in intricate layouts or more often in image format.

- Proposed Feature: Increase the scope of document compatibility to include academic documents having heavy layouts and image formats, broadening the system’s utility across more complex and realistic admissions scenarios.
- Technical Implementation: Use pdf2image to convert PDFs into images for layout analysis. Implement LayoutParser or pdfplumber for extracting content blocks from structured layouts. Apply image preprocessing (e.g., denoising, contrast enhancement via OpenCV) for OCR accuracy. Classify document types using scikit-learn or lightweight transformers to apply tailored parsing paths. Add XML/HTML parsing support for structured formats like Europass CVs.

9.4 Exportable Upload History and Session Reports

The existing Upload History panel gives users visibility into previously processed documents. However, there is value in allowing users to export session summaries for internal documentation, auditing, or collaboration.

- Proposed Feature: Enable exporting a session report listing all uploaded files, extracted institutions, and matched rankings supporting accountability, team workflows, and batch decision review.
- Technical Implementation: Store session records in a pandas DataFrame in memory. Use DataFrame.to_csv() or to_excel() to export reports with customisable fields. Include timestamp, file name, detected institution, ranking tier, and ranking source. Add a Flask-based download endpoint or direct frontend link to download reports. Ensure all data remains local and is purged on session end, unless explicitly saved.

9.5 Optional Extraction of Applicant Names for Enhanced Organisation

In its current version, the application intentionally avoids extracting personal identifiers such as applicant names. However, allowing name extraction—when explicitly enabled by the user—could facilitate file organisation, filtering, and reporting.

- Proposed Feature:

Extract applicant names using Named Entity Recognition (NER), and allow them to be included in session history or export reports, balancing operational convenience with ethical data practices, and enhancing usability for high-volume admissions workflows.

- Technical Implementation:

Use LLaMA (if supported) or integrate spaCy NER to extract PERSON entities. Allow users to enable this feature via a frontend toggle. Display names in the History UI and attach them to export summaries. Maintain data minimisation by keeping name data in-memory only, unless explicitly saved.

9.6 Integration with Admissions Systems

Currently standing as an independent local tool, the Institution Ranking Checker, having its results integrated into and interfaced with the greater admissions infrastructure of Trinity Business School, would go a strong way toward enhancing workflow continuity and achieving administrative efficiency.

- Proposed Feature:

Set up interfaces with internal admissions systems such as CRM systems, applicant tracking systems, or review dashboards through either structured output formats or direct data interface.

- Technical Implementation:

Allow batch export of processed results in standardised formats such as CSV, Excel, or JSON, with schema fields aligned to internal system requirements (e.g., applicant ID, institution match, ranking tier, timestamp). Develop export templates (e.g.,

TBS_Review_Format.xlsx) that map to specific workflows, such as faculty review or scholarship evaluation. Optionally add a configurable output folder path, where processed results can be automatically saved for syncing with shared drives or internal upload portals. For future extensibility, build a lightweight API layer using Flask-Restful or FastAPI that could, if needed, expose selected endpoints for internal use within a fully secure and offline-controlled network. Include applicant-level metadata in export only if name extraction (9.5) is enabled, and ensure no sensitive document content is transferred—only processed output. This pathway, on the other hand, ensures that the Institution Ranking Checker is not a standalone tool but rather a modular component nicely fitting into the existing digital admissions ecosystem at Trinity, thereby eliminating duplication of effort and increasing the reviewer efficiency, which in turn increases institutional assessment, thus fostering stakeholder consistency.

Chapter 10: Conclusion

The Institution Ranking Checker marks a meaningful step forward in bridging the gap between manual admissions processing and intelligent, automated institutional ranking evaluation. By leveraging both traditional fuzzy matching and AI-driven techniques within a flask based, secure, local-first framework, the system provides a practical, GDPR-compliant tool tailored to the operational realities of the Trinity Business School. Its ability to extract and match institutions from a wide variety of documents, while preserving user control and data privacy, reflects a thoughtful balance between technical capability and administrative need.

While the current implementation has its limitations—such as OCR sensitivity and large language model's latency—the system lays a solid foundation for more scalable, flexible, and internationally relevant applications. With planned enhancements like dynamic data updates, multilingual support, and potential integration into broader admissions ecosystems, the Institution Ranking Checker is well-positioned to evolve into a powerful tool for modern academic evaluation. To sum up, this project reflects how focused technological innovation, grounded in real user workflows, can make a meaningful impact in higher education administration.

Bibliography

AEPD (2021) *Risk Management and Impact Assessment in the Processing of Personal Data*. Available at: https://www.aepd.es (Accessed: 21 July 2025).

Amazon AWS. (2020). *Textract Developer Guide*. https://docs.aws.amazon.com/textract/

Bass, L., Clements, P., & Kazman, R. (2012). *Software Architecture in Practice* (3rd ed.). Addison-Wesley.

Christopher Kuner, Fred H. Cate, Christopher Millard, Dan Jerker B. Svantesson, Orla Lynskey, Risk management in data protection, *International Data Privacy Law*, Volume 5, Issue 2, May 2015, Pages 95–98, https://doi.org/10.1093/idpl/ipv005

EDPS (2018) *Opinion 5/2018 on Privacy by Design*. European Data Protection Supervisor. Available at: https://edps.europa.eu (Accessed: 21 July 2025).

Fowler, M. (2004). *Patterns of Enterprise Application Architecture*. Addison-Wesley.

Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.

Hevner, A., & Chatterjee, S. (2010). *Design Research in Information Systems: Theory and Practice*. Springer.

Li, J., Yu, K., Li, W., & Zhou, X. (2020). Smart Autocomplete Techniques in Structured Data Search. *IEEE Transactions on Knowledge and Data Engineering*, 32(10), 2032–2046.

Loukides, M. (2011). *What is Data Science?*. O'Reilly Media.

Meta AI. (2023). *LLaMA: Open Foundation and Fine-Tuned Chat Models*. https://ai.meta.com/llama/

McKinney, W. (2010). Data Structures for Statistical Computing in Python. *Proceedings of the 9th Python in Science Conference*, 51–56.

NIST (2017) *NISTIR 8062: An Introduction to Privacy Engineering and Risk Management in Federal Systems*. Gaithersburg, MD: National Institute of Standards and Technology.

NIST (2020a) *SP 800-37 Revision 2: Risk Management Framework for Information Systems and Organizations*. Gaithersburg, MD: National Institute of Standards and Technology.

NIST (2020b) *SP 800-53 Revision 5: Security and Privacy Controls for Information Systems and Organizations*. Gaithersburg, MD: National Institute of Standards and Technology.

Seiffert, M. (2021). *RapidFuzz: Fuzzy String Matching in Python*. https://github.com/maxbachmann/RapidFuzz

Smith, R. (2007). An overview of the Tesseract OCR engine. *Proceedings of the Ninth International Conference on Document Analysis and Recognition*.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M. A., Lacroix, T., ... & Scialom, T. (2023). LLaMA: Open and Efficient Foundation Language Models. arXiv preprint arXiv:2302.13971.

Appendix 1: Code GitHub Repository

The complete source code for this project can be accessed via the following GitHub repository.

Appendix 2: Fuzzy Matching Method Test Records

1. ID: 19326759

- Manual Verification: Amity University, Noida.
- Tool Extracted Output: "Amity University" (correct)
- Notes: Location and degree context aided correct match.

2. ID: 20301924 (UK University)

- Manual Verification: University of Hertfordshire
- Tool Extracted Output: University of Hertfordshire
- Notes: The current analysis is restricted to Indian universities; institutions from the UK were excluded from consideration.

3. ID: 20303929 (Canadian University)

- Manual Verification: The University of British Columbia
- Tool Extracted Output: The University of British Columbia
- Notes: The current analysis is restricted to Indian universities; the Canadian university were excluded from consideration.

4. ID: 23347107

- Manual Verification: BNM Institute of Technology (affiliated with Visvesvaraya Technological University).
- Tool Extracted Output: Visvesvaraya Technological University (correct)
- Notes: Affiliation correctly recognised despite poor OCR.

5. ID: 25333210

- Manual Verification: Veermata Jijabai Technological Institute (VJTI), University of Mumbai
- Tool Extracted Output: University of Mumbai (correct)
- Notes: Parent university correctly extracted.

6. ID: 25333924

- Manual Verification: Vivekanand Education Society's Institute of Technology (VESIT), Mumbai, India (University of Mumbai)
- Tool Extracted Output: Birla Institute of Technology (incorrect)
- Notes: Incorrect match despite correct university being present in text.

7. ID: 25334850

- Manual Verification: Two different universities on documents (savitribai phule pune university on transcript, Ajeenkya D.Y Patil School Of Engineering on CV and LOR)
- Tool Extracted Output: Savitribai Phule Pune University (correct)
- Notes: Extracted from correct source (transcript).

8. ID: 25335005

- Manual Verification: Savitribai Phule Pune University
- Tool Extracted Output: Savitribai Phule Pune University (correct)

9. ID: 24378480

- Manual Verification: University of Mumbai
- Tool Extracted Output: University of Mumbai (correct)
- Notes: Extracted correctly despite the presence of secondary school transcripts.

10. ID: 25332126

- Manual Verification: Bengaluru City University
- Tool Extracted Output: N/A (OCR Failure)
- Notes: OCR unable to extract all text.

11. ID: 25332331

- Manual Verification: Savitribai Phule Pune University)
- Tool Extracted Output: National College (Incorrect)
- Notes: Misled by school names in text.

12. ID: 25355469

- Manual Verification: Calcutta University
- Tool Extracted Output: N/A (OCR Failure)
- Notes: OCR unable to extract all text

13. ID: 25338504

- Manual Verification: University of Mumbai
- Tool Extracted Output: N/A (OCR Failure)
- Notes: School-level documents created noise.

14. ID: 25341865

- Manual Verification: Narsee Monjee Institute of Management Studies- NMIMS (present in dataset but not spotted due to overpowering “University” word weight)
- Tool Extracted Output: University of Delhi (Incorrect)
- Notes: When universities names are too short results in words like University giving fuzzy matching high scores.

15. ID: 25344164

- Manual Verification: University of Delhi

- Tool Extracted Output: University of Delhi (correct)

16. ID: 25351025
- Manual Verification: 2024 O.P. Jindal Global University- Sonipat, HR-131001, India
- Tool Extracted Output: Vellore Institute of Technology (Incorrect)
- Notes: University only mentioned in CV; not captured correctly.

17. ID: 23369257
- Manual Verification: Kurukshetra University
- Tool Extracted Output: Kurukshetra University (correct)

18. ID: 24333055
- Manual Verification: Anna University
- Tool Extracted Output: Anna University (correct)

19. ID: 24340591
- Manual Verification: HSNC University
- Tool Extracted Output: AU College of Engineering (A) (Incorrect)
- Notes: Institution not in ranking list; result unrelated.

20. ID: 25333091
- Manual Verification: devi ahilya vishwavidyalaya
- Tool Extracted Output: Devi Ahilya Vishwavidyalaya (correct)

21. ID: 25335257
- Manual Verification: Visvesvaraya Technological University, Belagavi
- Tool Extracted Output: Visvesvaraya Technological University (correct)

22. ID: 25337145
- Manual Verification: St. Thomas College, Kozhencherry
- Tool Extracted Output: N/A (Failure)
- Notes: University not present in Ranking list

23. ID: 25359387
- Manual Verification: apj abdul kalam technological university
- Tool Extracted Output: AU College of Engineering (A) (Failure)
- Notes: University not in ranking list but incorrectly returned results. Looks like due to dominant words such as college/engineering that are common words.

24. ID: 25359631
- Manual Verification: University of Delhi
- Tool Extracted Output: University of Delhi (correct)

25. ID: 25356151
- Manual Verification: St.Jospeh's engineering college, Visvesvaraya Technological University
- Tool Extracted Output: Visvesvaraya Technological University (correct)

26. ID: 25359189
- Manual Verification: SRM Institute of Science and Technology
- Tool Extracted Output: SRM Institute of Science and Technology (correct)

27. ID: 25359439
- Manual Verification: N.M.A.N Institute of Technology, Nitte (Autonomous), Udupi, Karnataka, India Affiliated to Visvesvaraya Technological University, Belagavi, Karnataka, India
- Tool Extracted Output: University of Science and Technology (Incorrect)
- Notes: Same as before too much heavy weighted common words

28. ID: 25359481
- Manual Verification: audyogik shikshan mandal (ASM)
- Tool Extracted Output: Savitribai Phule Pune University (Incorrect).
- Notes: University not in ranking list

29. ID: 25352131
- Manual Verification: St. Francis Institute of Technology, University of Mumbai
- Tool Extracted Output: Mumbai University (Incorrect)

30. ID: 25354803
- Manual Verification: St. Andrew's College of Arts, Science & Commerce
- Tool Extracted Output: Dr. N. G. P. Arts and Science College (Incorrect)
- Notes: Heavy weighted common words

31. ID: 25360626
- Manual verification: Sri Guru Gobind Singh College of Commerce, University of Delhi
- Tool Extracted Output: N/A (OCR Failure)

32. ID: 25362856
- Manual verification: Christ University
- Tool Extracted Output: N/A (Failure)

33. ID: 25367040
- Manual verification: H.R. College of Commerce and Economics, Mumbai
- Tool Extracted Output: N/A (Failure)

34. ID: 25368334

- Manual verification: University of Delhi
- Tool Extracted Output: University of Delhi (correct)

35. ID: 25369378
- Manual verification: R.V. College of Engineering
- Tool Extracted Output: Charotar University (Incorrect)

36. ID: 25369628
- Manual verification: Anna University
- Tool Extracted Output: Anna University (correct)

37. ID: 25370088
- Manual verification: Anil Surendra Modi School of Commerce (ASMSOC), NMIMS University
- Tool Extracted Output: N/A (failure)

38. ID: 25371646
- Manual verification: Maharshi Dayanand University
- Tool Extracted Output: N/A (failure)

39. ID: 25373208
- Manual verification: University of Mumbai
- Tool Extracted Output: Mumbai University (Incorrect)

40. ID: 25374012
- Manual verification: University of Pune
- Tool Extracted Output: N/A (OCR failure)

41. ID: 25374624
- Manual verification: Chennai Institute of Technology
- Tool Extracted Output: N/A (failure)

42. ID: 25375703
- Manual verification: Navrachana University
- Tool Extracted Output: N/A (OCR failure)

Appendix 3: LLaMA Large Language Model Test Records

1. 25354803
- Manual verification: St. Andrew's College of Arts, Science & Commerce
- Result returned: No match

2. 25355469
- Manual verification: Calcutta University
- Result returned: No match
- Notes: OCR unable to extract all text

3. 25356151
- Manual verification: St.Jospeh’s engineering college, Visvesvaraya Technological University
- Result returned: Visvesvaraya Technological University

4. 25359189
- Manual verification: Indian Institute of Technology Madras (IITM)
- Result returned: Indian Institute of Technology Madras (IITM)

5. 25359387
- Manual verification: apj abdul kalam technological university
- Result returned: Charotar University of Science & Technology (Incorrect)
- Note : Llama extracted: APJ Abdul Kalam Technological university, Kerala

6. 25359439
- Manual verification: NITTE
- Result returned: NITTE

7. 25359481
- Manual verification: audyogik shikshan mandal (ASM)
- Result returned: No match

8. 25359631
- Manual verification: University of Delhi
- Result returned: University of Delhi

9. 25360626
- Manual verification: Sri Guru Gobind Singh College of Commerce, University of Delhi
- Result returned: Sri Guru Gobind Singh College of Commerce

10. 25362856
- Manual verification: Christ University
- Result returned: Christ University

11. 25367040
- Manual verification: H.R. College of Commerce and Economics, Mumbai
- Result returned: No match

12. 25368334
- Manual verification: University of Delhi
- Result returned: University of Delhi

13. 25369378
- Manual verification: R.V. College of Engineering
- Result returned: Charotar University of Science & Technology (Incorrect)

14. 25369628
- Manual verification:Anna University
- Result returned: Anna University

15. 25370088
- Manual verification: Anil Surendra Modi School of Commerce (ASMSOC), NMIMS University
- Result returned: no match

16. 25371646
- Manual verification: Maharshi Dayanand University (MDU), Rohtak, Haryana, India
- Result returned: no match

17. 25373208
- Manual verification: University of Mumbai
- Result returned: no match

18. 25374012
- Manual verification: University of Pune
- Result returned: no match

19. 25374624
- Manual verification: Chennai Institute of Technology
- Result returned: Chennai Institute of Technology

20. 25375703
- Manual verification: Navrachana University

- Result returned: no match

21. 19326759
- Manual verification: Amity University
- Result returned: Amity University

22. 20301924
- Manual verification: University of Hertfordshire, UK
- Result: No institution was matched

23. 20303929
- Manual verification: The University of British Columbia
- Result: No institution was matched.

24. 23347107
- Manual verification: The University of British Columbia
- Result: No institution was matched.

25. 23369257
- Manual verification: Kurukshetra University
- Result: Kurukshetra University

26. 24333055
- Manual verification: Anna University
- Result: Anna University

27. 24340591
- Manual verification: KC College
- Result: University of Mumbai (Incorrect)

28. 24378480
- Manual verification: University of Mumbai
- Result: University of Mumbai

29. 25332126
- Manual verification: Presidency college Bangalore
- Result: Presidency College

30. 25332331
- Manual verification: INDIRA INSTITUTE OF MANAGEMENT (An Autonomous Institute affiliated to Savitribai Phule Pune University)
- Result: Savitribai Phule Pune University

31. 25333091

- Manual verification: university Devi Ahilya Vishwavidyalaya
- Result: Devi Ahilya Vishwavidyalaya

32. 25333210
- Manual verification: Veermata Jijabai Technological Institute(Affiliated to University of Mumbai)
- Result: University of Mumbai

33. 25333924
- Manual verification: Vivekananda Education Society's Institute of Technology(Affiliated to University of Mumbai)
- Result: University of Mumbai

34. 25334850
- Manual verification: Savitribai Phule Pune University
- Result: Savitribai Phule Pune University

35. 25335005
- Manual verification: Savitribai Phule Pune University
- Result: Savitribai Phule Pune University

36. 25335257
- Manual verification: Visvesvaraya Technological University
- Result: Visvesvaraya Technological University

37. 25337145
- Manual verification: Mahatma Gandhi University
- Result: Mahatma Gandhi University

38. 25338504
- Manual verification: DATTA MEGHE COLLEGE OF ENGINEERING
- Result: R.V. College of Engineering (Incorrect)

39. 25341865
- Manual verification: SVKM`s Narsee Monjee Institute of Management Studies
- Result: SVKM`s Narsee Monjee Institute of Management Studies

40. 25344164
- Manual verification: Jesus and Mary college, university of Delhi
- Result: University of Delhi

41. 25352131
- Manual Verification: St. Francis Institute of Technology, University of Mumbai
- Result: St. Francis Institute of Technology

42. 25351025

- Manual Verification: O.P. Jindal Global University
- Result: O.P. Jindal Global University