Admit-Guide: An End-to-End AI System for University Admission Guidance

CHATHUR BUJA RAM BATHI
University of Florida
Email: Cbathi@ufl.edu

***Abstract*—This paper presents *Admit-Guide*, an end-to-end AI system designed to support prospective graduate students by estimating admission likelihood and recommending suitable university programs in the United States. The system is built on a curated dataset of universities with structured admission requirements, including minimum CGPA, average GRE and TOEFL/IELTS expectations, acceptance rates, institutional ratings, and program strength areas. A hybrid methodology is adopted: a supervised classification pipeline predicts whether a candidate profile is a good match for a selected university, while a TF–IDF-based similarity module recommends programs aligned with a student's free-text interests. Logistic Regression and Random Forest models are developed as core classifiers; both achieve perfect accuracy (1.0) and F1-score (1.0) on the held-out evaluation split, and the Random Forest achieves a mean cross-validation accuracy of 0.8889 across three folds. Beyond raw performance, the project emphasizes robust AI lifecycle management, including data preprocessing, feature engineering, model selection, fairness analysis, explainability via SHAP, deployment using Docker, and monitoring with Prometheus and Grafana. A Gradio-based user interface supports two primary user journeys: admission probability estimation and program search, with integrated feedback collection to support continuous improvement. The results demonstrate that even a relatively small, domain-specific dataset, intentionally constructed rather than scraped from external sources, can be transformed into a transparent, monitored, and user-centered AI assistant when lifecycle, trustworthiness, and human-computer interaction principles are systematically integrated from problem definition through monitoring and maintenance.**

***Index Terms*—AI lifecycle management, university admissions, Random Forest, TF–IDF, explainability, fairness, Docker, Prometheus, Grafana, Gradio.**

I. Introduction

University admissions are a high-stakes decision process for both applicants and institutions. Applicants must navigate a complex landscape of programs, admission requirements, and implicit selectivity factors, often relying on fragmented sources such as forums, anecdotal advice, and partially documented university websites. For many first-generation or international students, this process can be opaque and anxiety-inducing, particularly when they are unfamiliar with the expectations of US graduate programs.

While universities commonly provide summary statistics—such as average admitted GRE scores or minimum GPA thresholds—these numbers are rarely integrated into a single decision-support environment. Moreover, they are almost never combined with a student's own academic interests (e.g., "machine learning for healthcare") in a way that can help generate a tailored set of program recommendations. As a result, applicants may under- or over-estimate their chances, apply to misaligned programs, or miss opportunities at institutions that would be a strong fit.

In addition, the admissions landscape is saturated with online content of varying quality, including blog posts, YouTube videos, informal spreadsheets, and community-maintained lists. These resources can be extremely helpful, but they are rarely organized around a coherent decision process or grounded in a clear model of uncertainty and risk. A system that can combine structured requirement data with user-facing explanations, monitoring, and explicit articulation of limitations therefore has the potential to reduce confusion and make planning more systematic.

This work introduces *Admit-Guide*, a prototype AI system that addresses two complementary questions for graduate applicants:

1) **Admission prediction**: "Given my CGPA and standardized test scores, how likely am I to satisfy the admission requirements of a specific university in the dataset?"
2) **Program recommendation**: "Given my high-level interests (e.g., 'machine learning', 'data science', 'cybersecurity'), which universities and programs in the dataset are most aligned with these interests?"

The primary goal is not only to build accurate predictive models but also to manage the entire AI lifecycle: from data collection and preprocessing, through model development and evaluation, to deployment, monitoring, and maintenance. The system is intentionally designed as a *trustworthy advisor* rather than a black-box oracle. In particular, the design emphasizes transparency, fairness considerations, monitoring of operational behavior, and a user-centered interface.

From a lifecycle perspective, *Admit-Guide* is treated as a small-scale but realistic case study: each stage (problem framing, data creation, modeling, deployment, monitoring, and improvement) is explicitly documented, and potential risks are identified and mitigated where possible. This provides a concrete example of how AI lifecycle management concepts can be instantiated in an educational project, and how even student projects can start from a governance and risk mindset.

The main contributions of this work are:

- A fully implemented AI system that integrates supervised admission prediction with TF–IDF-based program recommendation in a single Gradio-based interactive interface.

- A lifecycle-aware design that explicitly addresses data management, model development, deployment, monitoring, and maintenance, including risk and trustworthiness considerations at each stage.
- A practical deployment architecture based on Docker, Prometheus, and Grafana, enabling real-time monitoring of request volume, latency, and user feedback in a local multi-container environment.
- An initial fairness and model explainability analysis using state-wise accuracy and SHAP value visualizations, demonstrating how trustworthiness techniques can be applied even in modest student projects.

The remainder of this paper is organized as follows. Section II summarizes related work on admission prediction, educational recommender systems, lifecycle management, and MLOps. Section III describes the system architecture and the design of the data, model, deployment, and HCI components. Section IV presents the trustworthiness and risk management strategies applied at each lifecycle stage. Section V reports evaluation results, including predictive performance, fairness, explainability, and monitoring. Section VI discusses strengths, limitations, and broader implications. Section VII outlines future work, and Section VIII concludes the paper.

II. Related Work

A. Admission Prediction and Educational Analytics

Machine learning has been applied to student performance, dropout prediction, and admission decisions in several contexts. Classic treatments in machine learning textbooks [1] describe the use of regression and classification algorithms for predicting outcomes based on structured features such as GPA, test scores, and demographic variables. In educational data mining, models are often trained to predict student persistence, course success, or time to graduation rather than admission decisions.

Several works explore admission prediction using logistic regression, decision trees, and support vector machines, reporting promising accuracy when trained on institutional datasets. In such settings, the models may access proprietary admission records, enabling them to learn from thousands of historical decisions. However, these systems are typically embedded in internal decision workflows and are not exposed to students. Moreover, they rarely incorporate monitoring, trustworthiness analyses, or user-centered interfaces designed for applicants, and details about their performance and limitations can remain opaque.

Beyond direct admission prediction, there is a broader body of work on learning analytics and student success prediction. These studies often consider longitudinal data such as course grades over multiple semesters, participation data from learning management systems, and advising logs and use supervised learning to identify students at risk of dropping out or failing core courses. While technically related, these systems are usually designed for institutional administrators and advisors, rather than for students to use directly during the application phase.

B. Text-Based Recommendation and TF-IDF

TF–IDF is a classical technique for measuring the relevance of documents to a query by weighting terms based on frequency and specificity [2]. It remains competitive for retrieval tasks in domains where texts are relatively short and topical, such as document search or product descriptions. While modern recommender systems increasingly employ deep neural embeddings, TF–IDF is attractive in educational decision-support contexts because it is transparent, fast to compute, and easy to explain to non-experts.

In the context of program recommendation, TF–IDF can be applied to textual descriptions of programs, research areas, and institutional focus. Users can express their interests in natural language, and the system can return programs with high cosine similarity in the TF–IDF space. This approach is particularly suitable when datasets are small and when the priority is interpretability rather than incremental performance gains. It also avoids the need for expensive pre-training or fine-tuning of large language models, which may be impractical in course projects or resource-constrained deployments.

C. Model Evaluation, Fairness, and Explainability

Standard evaluation metrics such as accuracy, precision, recall, and F1-score are widely used to assess classification models [3]. However, aggregate metrics can mask disparities across subgroups, such as geographic regions or demographic categories. Fairness toolkits such as Fairlearn [7] and AIF360 [8] support disaggregated analysis and mitigation of algorithmic bias, providing metrics such as demographic parity difference, equalized odds, and subgroup-specific error rates.

Explainability methods such as LIME [5] and SHAP [6] provide local and global explanations for model predictions, respectively. LIME fits interpretable surrogate models around individual predictions, while SHAP is grounded in cooperative game theory and provides additive explanations that can be aggregated across samples to study feature importance. These techniques are increasingly viewed as essential in high-stakes decisions, including education, hiring, and healthcare, where stakeholders may require both performance and justification.

D. Lifecycle Management, Deployment, and Monitoring

MLOps and AI lifecycle management frameworks emphasize that model training is only one component of a production AI system [12]. Best practices highlight the importance of version control, reproducible builds, continuous integration and deployment (CI/CD), and real-time monitoring. Containerization with Docker and orchestration frameworks (e.g., Kubernetes) are commonly used to achieve reproducibility and scalability.

Prometheus and Grafana are widely used in cloud-native environments for metrics collection and visualization [9], [10]. In the context of AI systems, they can track not only infrastructure metrics (CPU, memory) but also application-level indicators such as request counts, latencies, error rates, and model-specific signals (e.g., distribution of predicted probabilities). Lightweight front-end libraries like Gradio [11] enable rapid development of ML-powered interfaces that can be easily demoed and shared, lowering the barrier for non-experts to interact with models.

E. Trustworthiness, Risk, and Governance

Recent guidelines such as the NIST AI Risk Management Framework (AI RMF) highlight the importance of addressing trustworthiness characteristics fairness, transparency, robustness, privacy, and accountability across the AI lifecycle [13]. Rather than treating these aspects as afterthoughts, the AI RMF recommends integrating them into problem definition, data management, model development, deployment, and monitoring stages.

The *Admit-Guide* system aligns with these recommendations by explicitly documenting risks, implementing some technical mitigations (e.g., fairness evaluation, explainability, monitoring), and treating residual risk assessment as part of the project deliverables. In addition, the project treats the educational context itself as a form of governance, where the goal is not just performance but also reflection on the consequences of deploying such systems in practice. This perspective is particularly relevant as universities begin to experiment with AI-powered advising tools that may influence students' major choices, course loads, and career trajectories.

Compared to prior work, *Admit-Guide* combines admission prediction, program recommendation, lifecycle-aware MLOps practices, and trustworthiness analyses into a single integrated system aimed at student-facing decision support.

III. System Design and Implementation

A. Overall Architecture

The system is organized into four main layers:

1) **Data layer**: a structured spreadsheet of university admission requirements and program descriptors.
2) **Modeling layer**: a scikit-learn pipeline for admission prediction and a TF–IDF engine for program similarity.
3) **Interface layer**: a Gradio-based web UI exposing two tabs: admission prediction and program finder.
4) **Operations layer**: Docker-based deployment, Prometheus metrics export, and Grafana dashboards.

Fig. 1 provides a high-level view of the workflow that connects these layers, from user input through computation to monitoring. This modularization also reflects the AI lifecycle stages: the data layer supports collection and preprocessing, the modeling layer enables training and evaluation, the interface layer supports HCI and user feedback, and the operations layer supports deployment, monitoring, and maintenance.

Text inputs from the user follow two distinct flows. In the admission prediction path, structured numeric features (scores, CGPA) and a categorical university identifier are mapped into feature vectors, transformed by the preprocessing pipeline, and passed into the trained classifier to produce probabilities and predicted labels. In the recommendation path, the query text passes through a TF–IDF vectorizer, and cosine similarities are computed between the query and all program descriptions stored in the dataset. The resulting scores are sorted and presented back to the user as a ranked list of programs.

Interest Text $\rightarrow$ TF–IDF Vector $\rightarrow$ Cosine Similarity $\rightarrow$ Ranked List of Programs(Program Finder Flow)

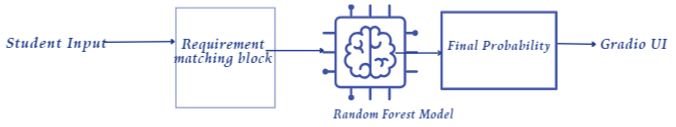


Fig. 1. High-level workflow of the Admit-Guide system, spanning data ingestion, model pipelines, the Gradio user interface, and monitoring with Prometheus and Grafana.

B. Dataset Construction and Curation

A key design choice in this project is that the dataset is *constructed* rather than scraped from a specific external source. Instead of copying a single university ranking list or proprietary admissions dataset, a synthetic but realistic dataset was manually designed to approximate plausible requirement ranges across a variety of US institutions.

The data creation process followed these steps:

- Identification of a diverse set of hypothetical universities and locations, inspired by typical US states and institution types (public/private, high-tier vs. mid-tier).
- Assignment of program strength areas (e.g., AI/ML, Data Science, Cybersecurity, Software Engineering) to each university, ensuring coverage of multiple computing subfields.
- Definition of admission requirement ranges for GRE, TOEFL/IELTS, CGPA, and acceptance rates based on publicly observable patterns (e.g., highly ranked schools tend to have higher GRE/CGPA expectations and lower acceptance rates).
- Validation that the resulting combinations are internally consistent (e.g., very high acceptance rates are not paired with extremely high requirements).

No single proprietary or copyrighted source is reproduced; instead, the dataset functions as a controlled, pedagogical approximation of the admissions landscape. This has several advantages:

- It avoids legal and ethical concerns associated with scraping and redistributing institutional data.
- It allows explicit control over feature distributions and relationships (e.g., ensuring variation in acceptance rates and requirements).
- It supports experimentation with synthetic labels, as discussed below, without implying that the model encodes real institutional behavior.

From a data lifecycle perspective, the manual construction process also makes it straightforward to document the assumptions behind each feature and to reason about how those assumptions might need to be revisited if the system were scaled up. For example, the current dataset does not model program-specific tuition or funding availability, but the schema is flexible enough to incorporate such fields in the future.

C. Label Definition and Synthetic Target

The target label (*Admission_Label*) is synthetically constructed to reflect whether a university is relatively accessible, based on thresholds over requirements and rating. Specifically, universities with moderate GRE and CGPA cutoffs, non-extreme ratings, and comparatively higher acceptance rates are labeled as more accessible (positive class), while highly selective universities with strict thresholds and low acceptance rates are labeled as less accessible (negative class).

This labeling scheme encodes an intuitive scenario:

- A student with average or slightly above-average credentials is more likely to gain admission at accessible universities.
- The model learns a notion of "difficulty" that correlates with these requirements and ratings, independent of any real historical outcomes.

The synthetic label has important implications for interpretation:

- It is *not* a proxy for real admission probabilities and must not be treated as such.
- It is suitable for demonstrating supervised learning, evaluation, fairness analysis, and explainability in a safe environment.
- It encourages users to view the system as a scenario-based advisor rather than as an oracle trained on confidential admissions data.

In practice, the label is generated by a set of hand-crafted rules that encode a simple scoring function over requirements and ratings. While this rule-based generator is not explicitly exposed to the user, it defines an underlying "ground truth" from which the learned models are allowed to deviate if they overfit or underfit. Comparing the rule-based scores with model predictions during development helped to verify that the training process was behaving as expected.

D. Data Schema and Preprocessing Pipeline

Table I summarizes the main structured features used for modeling.

TABLE I
KEY STRUCTURED FEATURES USED IN THE ADMISSION PREDICTION PIPELINE

Feature	**Description**
Average GRE Required	Typical GRE score expected for admission
Average TOEFL Required	Typical TOEFL score, where applicable
Average IELTS Required	Typical IELTS alternative
Minimum CGPA Required	Minimum GPA threshold
Acceptance Rate (%)	Historical acceptance rate (synthetic)
University Rating (1–5)	Overall institutional rating (synthetic)
University	Categorical identifier
Location (State)	US state of the institution
Program Strength Area	Primary specialization area

Preprocessing steps include:

- Handling missing values with simple imputation when necessary.
- Min–Max scaling for numerical features to the $[0, 1]$ range.
- One-hot encoding of categorical variables (university, state, program area).
- Construction of a textual *Program_Description* field by concatenating university, program area, and location, which is later embedded using TF–IDF for program search.

These steps are encapsulated in a scikit-learn ColumnTransformer combined with a Pipeline, ensuring that the same transformations are consistently applied during training and inference. Embedding the preprocessing logic inside the pipeline is also a key lifecycle practice: it reduces the risk of training/serving skew when the model is deployed and simplifies exporting the model as a self-contained artifact.

E. Model Development and Training

Two main models are developed for the admission prediction task.

1) Training Setup: The dataset is split into training and test sets using a standard hold-out strategy, with approximately 70% of the examples used for training and 30% reserved for evaluation. Within the training set, 3-fold cross-validation is applied to the Random Forest model to estimate generalization performance across different partitions. All experiments are run on a CPU-only environment, and training times are well under a second for each model, reflecting the modest size of the dataset.

2) Logistic Regression Baseline: Logistic Regression is selected as a simple, interpretable baseline model [1]. It operates on the preprocessed numerical and one-hot encoded categorical features. The model provides calibrated probabilities and allows inspection of learned coefficients, which can be mapped back to human-understandable features.

On the held-out test set, the Logistic Regression model achieves an accuracy of 1.0 and F1-score of 1.0, with a perfect confusion matrix (no misclassifications). The small dataset makes overfitting a concern; however, this model serves primarily as a transparent baseline against which more complex models can be compared. In principle, regularization strength could be tuned via cross-validation, but in this project the focus is on comparing logistic regression qualitatively with Random Forests and on using logistic regression as a sanity check for the data processing pipeline.

3) Random Forest with Structured Features: A Random Forest classifier [4] is trained using the same structured features. Random Forests can capture non-linear interactions between features and are robust to feature scaling choices. In this project, the Random Forest model also achieves test accuracy and F1-score of 1.0. To assess generalization, 3-fold cross-validation is conducted, yielding a mean accuracy of 0.8889, indicating that performance remains high across folds despite the limited sample size.

Hyperparameters such as the number of trees, maximum depth, and minimum samples per leaf are kept relatively modest to avoid overfitting and to keep the model interpretable.

While extensive hyperparameter tuning is possible, the emphasis here is on connecting the model to the rest of the lifecycle (deployment, monitoring, fairness analysis) rather than squeezing out marginal performance gains.

Fig. 2 shows the per-fold cross-validation accuracy for the Random Forest model, illustrating that performance is consistently strong, with some variability due to the small number of samples in each fold.

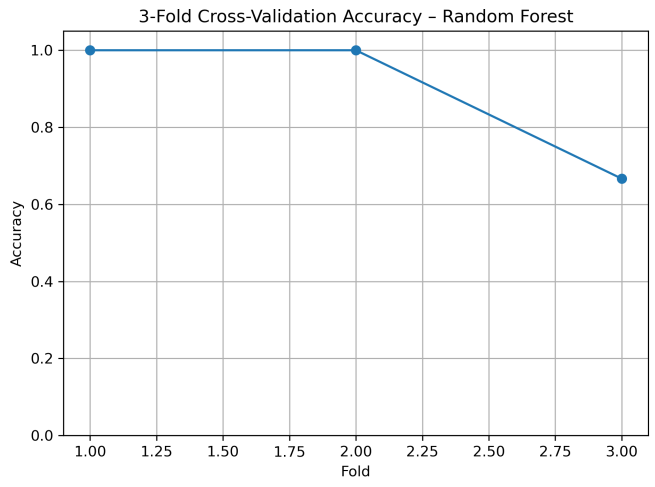


Fig. 2. Per-fold cross-validation accuracy for the Random Forest classifier, with a mean accuracy of 0.8889 across three folds.

The confusion matrix on the held-out test set is shown in Fig. 3. It confirms that both positive and negative classes are correctly predicted in this evaluation, reflecting that the synthetic label is easily learnable from the available features.

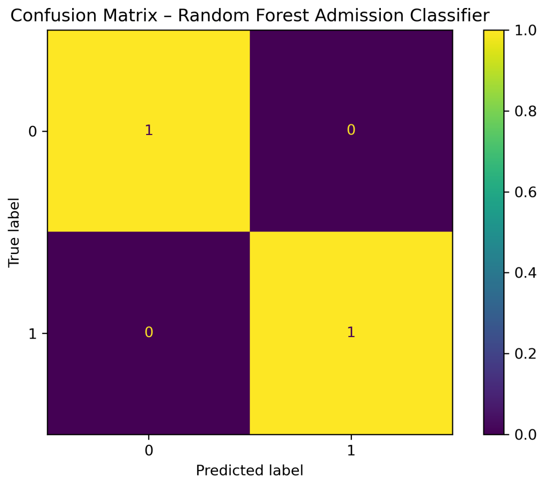


Fig. 3. Confusion matrix for the Random Forest classifier on the test split, showing perfect classification of both classes.

F. Explainability with SHAP

To understand feature contributions to the Random Forest model, SHAP (SHapley Additive exPlanations) values are computed [6]. SHAP assigns an importance value to each feature for each prediction, based on how much that feature contributes to deviating from a baseline prediction.

The global SHAP summary plot in Fig. 4 highlights the relative influence of features such as minimum CGPA requirement, average GRE, and university rating. This helps users and stakeholders understand which factors most strongly drive the model's decision to classify a university as “accessible” or “difficult”.

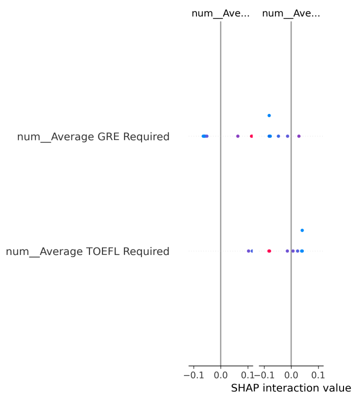


Fig. 4. SHAP summary plot for the Random Forest model, showing feature importance and the direction of impact on the prediction across the dataset.

From a lifecycle perspective, SHAP serves two roles: it is a debugging tool during model development (e.g., to detect unintuitive dependencies) and a documentation artifact that can be referenced when explaining system behavior to non-technical stakeholders. For example, if SHAP analysis revealed that the model placed excessive weight on a single university identifier, this could signal potential overfitting or a data encoding issue that would need to be addressed before deployment.

In addition to global explanations, local SHAP values can be computed for individual predictions and shown to users as part of the interface (e.g., explaining why a specific university is considered more or less accessible for a given profile). While this capability is not fully integrated into the current Gradio UI, the underlying code to compute local SHAP values is in place.

G. Program Recommendation via TF-IDF

The recommendation component is based on TF-IDF vectorization of the *Program_Description* field [2]. At query time, the user provides a short free-text description of their interests (e.g., “deep learning for healthcare” or “data analytics for business”). The text is vectorized, and cosine similarity is computed between the query vector and all program description vectors.

Top-ranked universities and program areas are returned in a table, enabling users to discover institutions whose focus areas align with their interests. In earlier iterations, numeric filters (e.g., minimum CGPA, GRE) were applied, but the final user-facing version emphasizes a simple interest based search interface that keeps the cognitive load low.

This approach has several advantages: it is transparent (the relevant terms can be inspected), efficient for small datasets, and easy to update when new programs are added. Because the TF-IDF model is based solely on institutional descriptors, it does not use any sensitive user data, which simplifies trust and privacy considerations at this stage of the project. The recommendation component, therefore, introduces minimal additional risk relative to the classification component, while still adding substantial value for exploration.

H. Deployment Strategy

The deployment is implemented as a local multi-container setup using Docker Compose. The following services are defined:

- **Application container**: Runs the Gradio interface and the ML pipelines on top of Python and scikit-learn. It exposes port 7860 for the UI and port 8000 for Prometheus metrics.
- **Prometheus container**: Scrapes metrics from the application at the metrics endpoint at a defined interval (e.g., every 5 seconds).
- **Grafana container**: Connects to Prometheus as a data source and renders dashboards that visualize application-level metrics.

The application image is built from a lightweight python:3.11-slim base image. The requirements.txt file specifies dependencies such as pandas, scikit-learn, gradio, prometheus-client, and openpyxl (for reading Excel files). By isolating the application environment from the host, Docker improves reproducibility and simplifies sharing the system with others.

Although a full CI/CD pipeline is beyond the scope of this project, the Docker-based structure makes it straightforward to later integrate automated builds and deployments using GitHub Actions or GitLab CI. From a lifecycle standpoint, containerization ensures that the deployed environment closely matches the development environment, reducing environment-related bugs and simplifying maintenance.

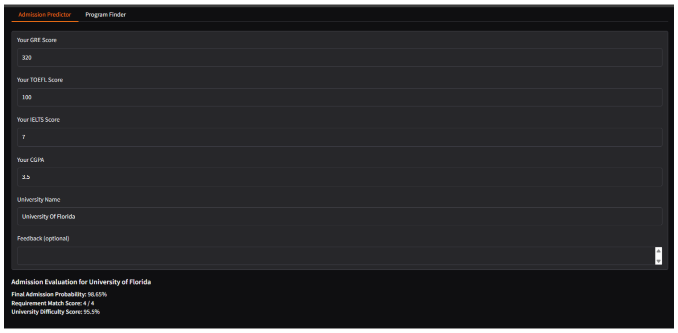
I. HCI Considerations and User Interface

The user interface is implemented using Gradio [11], which allows defining interactive components in Python and automatically generates a web front-end. Two tabs separate the primary tasks:

- **Admission Predictor**: Users enter their GRE, TOEFL/IELTS, CGPA, and the target university name. The system returns an estimated admission probability, a requirement match score, and textual explanations of how scores compare with university thresholds.
- **Program & University Finder**: Users describe their interests in a text box, and the system returns a table of matching universities and program areas based on TF–IDF similarity.

Early low-fidelity wireframes were used to explore layout options, including where to place explanatory text, how to group input fields, and how to present results. Feedback from self-testing emphasized the importance of clear labels, reasonable default values, and explanatory help text. For example, placeholder text is used to suggest example queries (e.g., "data science for finance"), and constraints on numeric inputs are added to prevent obviously invalid entries.

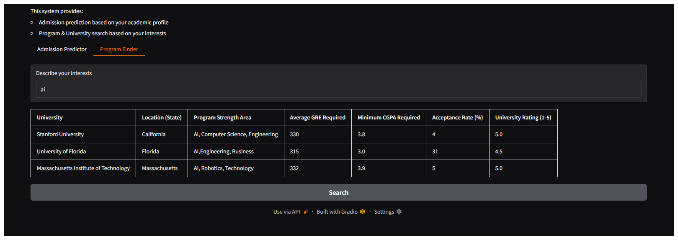
Fig. 5 and Fig. 6 show the final deployed Gradio interfaces.



Your GRE Score
320
Your TOEFL Score
100
Your IELTS Score
7
Your CGPA
3.5
University Name
University Of Florida
Feedback (optional)

Admission Evaluation for University of Florida
Final Admission Probability: 98.65%
Requirement Match Score: 4 / 4
University Difficulty Score: 95.5%

Fig. 5. Gradio user interface for the Admission Prediction module. Users input GRE/TOEFL/IELTS scores, CGPA, and university name to receive real-time predictions and explanations.



This system provides:
- Admission prediction based on your academic profile
- Program & University search based on your interests

Describe your interests
ai

University	Location (State)	Program Strength Area	Average GRE Required	Minimum CGPA Required	Acceptance Rate (%)	University Rating (1-5)
Stanford University	California	AI, Computer Science, Engineering	330	3.8	4	5.0
University of Florida	Florida	AI,Engineering, Business	315	3.0	31	4.5
Massachusetts Institute of Technology	Massachusetts	AI, Robotics, Technology	332	3.9	5	5.0

Fig. 6. Gradio user interface for the Program Recommendation module, enabling users to enter free-text academic interests and receive a ranked list of matching university programs.

The interface also includes a free-text feedback field, which is logged server side. This supports long-term improvement of the system by capturing user impressions of relevance and accuracy and is a first step toward incorporating human-in-the-loop feedback into the lifecycle. From an HCI perspective, the system emphasizes clarity, responsiveness, and the communication of uncertainty, rather than attempting to fully automate decision-making.

IV. Trustworthiness and Risk Management

Trustworthiness and risk management are considered at each stage of the AI lifecycle, following the spirit of the NIST AI RMF [13]. While the project is small in scale, the goal is to practice thinking about risks early, rather than retrofitting trust considerations after deployment.

A. Problem Definition

At the problem-definition stage, two primary risks were identified:

- **Misinterpretation risk**: Users might interpret the system's output as a guarantee of admission rather than a rough indication based on requirements.

- **Scope creep**: The system might be assumed to account for qualitative factors (e.g., personal statements, recommendation letters) that are not modeled.

To mitigate these risks, the system is explicitly framed as a *decision-support* tool. Disclaimers are included in the interface and documentation, clarifying that the system uses a limited dataset and synthetic labels and that real admissions decisions depend on many additional factors. The wording of the UI avoids terms like “guaranteed” or “will be admitted” and instead uses phrases such as “estimated accessibility” or “alignment with requirements.”

B. Data Collection and Management

At the data stage, the following risks are considered:

- **Data quality risk**: Inaccurate or outdated university requirement values.
- **Coverage risk**: Limited number of universities and program areas, potentially leading to biased coverage of the broader landscape.
- **Privacy risk**: Although the dataset contains institutional rather than personal data, any future extension involving user profiles must handle privacy carefully.

Mitigations include manual validation of the dataset, storing the spreadsheet under version control, and documenting data provenance. For this project, no personal data is collected beyond free-text feedback, which is stored without user identifiers. The decision to create a synthetic dataset further reduces privacy risks at this stage and simplifies compliance with data protection regulations that would otherwise apply to real applicant data.

C. Model Development Risks

Key risks during model development include:

- Overfitting due to small sample size, especially when models achieve perfect accuracy on a small test split.
- Spurious correlations between categorical identifiers (e.g., university name) and the synthetic label.
- Lack of robustness to future changes in admission practices or expanded data.

Mitigation strategies:

- Use of cross-validation to assess generalization beyond a single train/test split.
- Preference for relatively simple, well-understood models (Logistic Regression, Random Forest) rather than deep networks.
- Use of SHAP to inspect feature importance and verify that model decisions are aligned with domain intuition (e.g., higher minimum CGPA making universities less accessible).

These mitigations are not exhaustive, but they demonstrate a disciplined approach to model development that goes beyond simply maximizing accuracy.

D. Fairness and Group-Based Evaluation

Fairness is evaluated along the *Location (State)* attribute. As a preliminary analysis, the accuracy of the classifier is computed separately for California and Illinois. Fig. 7 shows that both groups achieve perfect accuracy on the small test split, leading to a minimum group accuracy of 1.0 and zero disparity in this setting.

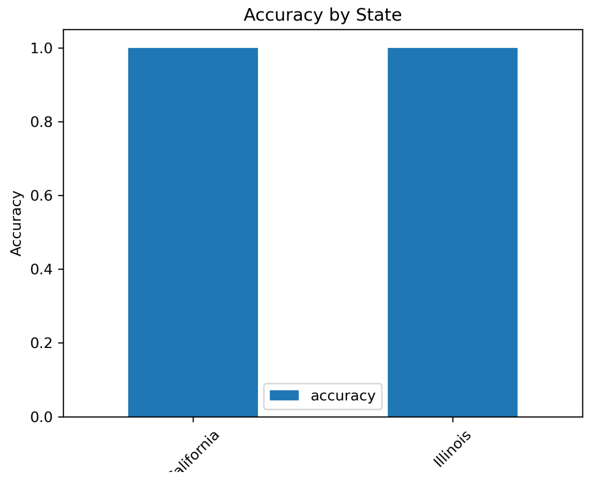


Fig. 7. Fairness evaluation by state: per-group accuracy for California and Illinois. In this small sample, both groups achieve accuracy of 1.0, though the sample size is limited.

However, this analysis is explicitly recognized as a proof of concept rather than a definitive fairness guarantee: the dataset is small and does not cover sensitive attributes such as race, gender, or socioeconomic status. In future, more comprehensive fairness evaluations could be implemented using toolkits like Fairlearn [7] and AIF360 [8], and fairness metrics such as demographic parity difference or equalized odds could be computed across protected groups. Fairness-aware interventions, such as post-processing of decision thresholds or reweighting of training examples, could then be explored if disparities are identified.

E. Deployment and Operational Risks

Deployment-related risks include:

- **Service availability**: Container crashes or misconfigurations could make the system temporarily unavailable.
- **Latency and resource usage**: Under heavier loads, response times could increase or memory could be exhausted.
- **Metrics exposure**: Exposing internal metrics publicly without authentication could leak implementation details.

Mitigation measures:

- Containerization with Docker and explicit configuration of dependencies to improve reproducibility.
- Restricting the deployment to a local or controlled environment in this project phase.

- Exporting only aggregated, non-PII metrics (request counts, latency histograms, feedback counts) via Prometheus.

While the current deployment does not include authentication or rate limiting, these controls would be important in a real-world deployment that serves external users over the public internet.

F. Residual Risk Assessment

A qualitative likelihood-impact matrix was constructed as part of the risk management plan. Examples of residual risks include:

- **Model drift**: As universities update their requirements over time, the static dataset becomes stale (possible likelihood, moderate impact).
- **User over-reliance**: Users might over-trust the predictions (possible likelihood, high impact).

Residual risks are managed through clear disclaimers, documentation, and the possibility of retraining the model when updated data becomes available. In a production environment, automated data pipelines and periodic retraining would be needed, along with stronger governance processes for validating changes and for approving model updates that could affect user decisions.

V. Evaluation and Results

A. Predictive Performance

Table II summarizes key performance metrics for the Logistic Regression and Random Forest models on the held-out test set. Both models achieve accuracy and F1-score of 1.0. The Random Forest model achieves a mean cross-validation accuracy of 0.8889 across three folds.

TABLE II
Performance Metrics on Test Set and Cross-Validation

Model	Accuracy	F1-score	CV Mean Acc.
Logistic Regression	1.00	1.00	–
Random Forest	1.00	1.00	0.89

The classification reports for both models indicate perfect precision and recall for both classes in the test set. Given the small size of the evaluation split, these results are interpreted cautiously: they demonstrate that the models fit the synthetic label well but do not guarantee performance on real-world admission data. Nonetheless, they confirm that the pipeline is internally consistent and that the feature set is sufficient to reconstruct the underlying rule-based labeling scheme.

B. Explainability and Feature Contributions

The SHAP summary plot (Fig. 4) reveals that the most influential features include minimum CGPA requirement, average GRE, and university rating. These align with domain expectations: more accessible universities tend to have lower minimum CGPA and GRE requirements and moderate ratings. Such alignment between model behavior and domain knowledge increases trust in the model, even in the absence of real admissions labels.

Qualitative inspection of local SHAP explanations further shows that, for individual predictions, the model tends to reduce predicted accessibility when minimum CGPA or GRE requirements are high, and to increase accessibility when acceptance rates are relatively high. This is consistent with the intended semantics of the synthetic label and helps build confidence that the model is learning meaningful patterns rather than arbitrary correlations.

C. Fairness Evaluation

As described in Section IV, fairness is evaluated along the state dimension. While the initial analysis suggests no disparity between California and Illinois in this dataset, the small sample size and synthetic target limit the conclusions that can be drawn. Nonetheless, incorporating fairness evaluation into the workflow demonstrates how similar methods could be applied when richer data becomes available and additional attributes (e.g., institution type, tuition range) are considered.

In a more realistic deployment, fairness analysis would likely focus on applicant level attributes such as gender, race, or socioeconomic status, which are not modeled here. The current setup can be viewed as a technical rehearsal for those more extensive analyses.

D. Program Search Evaluation

Program search quality is evaluated qualitatively by issuing several interest queries (e.g., “machine learning”, “data analytics”, “cybersecurity”) and inspecting the top-ranked results. In most cases, the returned universities and program strength areas align with expectations, confirming that the TF-IDF representation of program descriptions is capturing relevant topical information.

In addition, the diversity of recommendations is inspected: for broad queries like “computer science”, the system returns programs across multiple states and institution types, rather than concentrating solely on a single university. This suggests that the term distributions in the constructed dataset are reasonably balanced, although more systematic diversity metrics could be explored in future work.

Future quantitative evaluation could ask users to rate the relevance of recommendations or compare TF–IDF against embedding-based retrieval approaches. Such studies would provide more systematic evidence of the recommendation component’s effectiveness.

E. Monitoring Metrics and Feedback

Prometheus collects the following application level metrics:

- Total request count per endpoint (admission prediction vs. program search).
- Response time histograms per endpoint.
- Count of feedback submissions.

These metrics are visualized in Grafana dashboards over time. During manual testing, the system exhibits low latency (typically well under one second per request) and stable operation. Feedback entries captured via the UI can later be analyzed to understand user satisfaction and to prioritize improvements.

From a lifecycle perspective, monitoring provides observability into the system's behavior and forms the basis for future alerting and automated remediation workflows (e.g., detecting sustained increases in latency or error rates). Even in this small scale project, the experience of configuring and using monitoring tools is valuable preparation for larger deployments.

VI. Discussion

A. Strengths

The development of *Admit-Guide* highlights several technical and conceptual strengths that demonstrate the value of lifecycle aware AI system design. First, the project shows that even a relatively small, structured dataset—when carefully curated and systematically preprocessed—can be sufficient to build a meaningful prototype capable of supporting multiple user-facing tasks. By integrating admission prediction and program recommendation into one interface, the system provides an initial blueprint for holistic decision support tools in the education domain.

Second, the project illustrates that full stack AI system development is feasible in a course environment when supported by modern tools. The combined use of scikit-learn for modeling, Gradio for interface development, and Docker for containerization provides a lightweight ecosystem that mimics real-world ML operations. The inclusion of Prometheus for metrics scraping and Grafana for visualization demonstrates that monitoring, observability, and performance tracing often overlooked in academic prototypes can be integrated without heavy engineering overhead.

Third, the deliberate construction of a synthetic dataset highlights a responsible alternative to scraping real admissions data, which may introduce ethical concerns, privacy risks, or licensing restrictions. The synthetic dataset allows safe experimentation with fairness and explainability while ensuring that no identifiable student or institutional data is used. This creates a controlled environment where model behavior can be examined in depth without unintended real-world consequences.

Finally, the system emphasizes usability and user experience, providing a user friendly entry point for interacting with ML models. By offering clear explanations, structured feedback pathways, and intuitive interface navigation, *Admit-Guide* demonstrates how HCI considerations can be embedded early in the AI lifecycle to promote trust and user engagement.

B. Threats to Validity

Despite these strengths, several limitations must be recognized to correctly interpret the findings and scope of this project.

Internal validity is constrained by the synthetic nature of the target label. Because the admission outcome is generated through rule-based heuristics, the models are effectively learning deterministic patterns embedded in the data. While this is sufficient for demonstrating lifecycle management and system integration, it limits the interpretability of model behavior with respect to real admissions scenarios. Moreover, perfect accuracy on a test set in small datasets frequently indicates overfitting, reinforcing the caution needed when interpreting results.

External validity is similarly limited. The dataset includes a modest number of universities and program descriptions, and the distributions of GRE, TOEFL, CGPA, and acceptance rates are designed to be plausible rather than empirically validated. As a result, the findings cannot be generalized to the broader landscape of U.S. graduate admissions. Real admissions processes involve numerous qualitative factors letters of recommendation, personal statements, research fit, institutional priorities that are not represented here.

Construct validity poses another challenge. The proxy label "accessibility" is only one interpretation of institutional selectivity. Alternative constructs such as program competitiveness, faculty alignment, or funding availability may produce very different models and user experiences. Thus, while the label serves an instructional purpose, it is not a comprehensive representation of the admissions ecosystem.

Explicitly identifying these limitations ensures that the system is interpreted as a learning artifact and prototype rather than a deployable admissions tool. Acknowledging them also highlights the importance of rigorous dataset design, validation procedures, and domain knowledge when transitioning from a pedagogical prototype to a real-world AI system.

C. Broader Implications

Despite the constraints noted above, *Admit-Guide* has broader implications for both AI education and the development of institutional decision support systems. The project showcases a modular architecture that can be adapted to numerous domains beyond admissions. Any environment in which structured decision rules coexist with textual descriptions such as scholarship allocation, job matching, course planning, or internship selection could benefit from a similar hybrid modeling approach.

Moreover, the project serves as a practical demonstration of how responsible AI practices can be taught and applied. Students often learn about fairness, explainability, and monitoring in isolation; this system shows how these concepts integrate into a unified pipeline. By giving learners direct exposure to real deployment tools such as Docker and Prometheus, the project bridges the gap between ML coursework and real-world engineering practices.

The system also raises important discussions around the ethical use of AI in education. As universities increasingly explore automated advising tools, it is essential to understand the risks of misinterpretation, overreliance, and algorithmic bias. Prototypes like *Admit-Guide* can provide decision makers and researchers with insights into how such tools might be responsibly deployed, monitored, and governed.

Finally, the broader implication of this work is the growing importance of interdisciplinary thinking in AI system design. Successful deployment of trustworthy AI requires not only technical ability but also awareness of policy guidelines, user experience design, data governance, and continuous monitoring practices. By addressing these elements together, this project reflects the emerging standards for responsible AI systems and highlights how academic prototypes can meaningfully contribute to these conversations.

VII. Future Work and Improvements

Several directions can extend this work:

- **Richer data**: Incorporate real historical admissions outcomes and additional features such as statement-of-purpose embeddings, research experience, or work history, following appropriate privacy safeguards and governance.
- **Advanced embeddings**: Replace or augment TF–IDF with transformer-based sentence embeddings (e.g., SBERT) for more nuanced program similarity, while preserving some level of interpretability via keyword highlighting or exemplar-based explanations.
- **Automated drift detection**: Integrate drift detection tools (e.g., NannyML) to monitor changes in input distributions or model performance over time and trigger retraining workflows.
- **User studies**: Conduct usability evaluations (e.g., System Usability Scale) with students and advisors to refine the interface, explanations, and feedback mechanisms and to better understand how the tool is used in real decision-making contexts.
- **Cloud deployment**: Deploy the system to a cloud platform (e.g., AWS or GCP) with managed Kubernetes or serverless endpoints for scalability, resilience, and easier collaboration, and integrate authentication and role-based access control.

In addition, more sophisticated fairness and accountability mechanisms could be introduced, such as routine fairness audits, model cards describing limitations and suitable use cases, and governance processes for approving major changes to the system.

VIII. Conclusion

This paper presented *Admit-Guide*, a lifecycle-aware AI system for university admission prediction and program recommendation. The system combines structured requirement data with TF–IDF-based program descriptions, supervised machine learning models for accessibility classification, and an interactive Gradio-based user interface. A complete deployment pipeline—built with Docker, Prometheus, and Grafana demonstrates how monitoring, observability, and operational control can be integrated even in small-scale academic AI projects. Initial fairness checks and SHAP-based explainability analyses further illustrate how trustworthiness principles can be operationalized in an end-to-end system.

Beyond its technical implementation, *Admit-Guide* serves as a case study in responsible AI lifecycle management. By intentionally constructing a synthetic dataset, documenting assumptions, and restricting the scope of predictions, the project emphasizes safe experimentation without relying on sensitive or proprietary data. The work also highlights how important it is to accompany predictive models with guardrails, such as clear disclaimers, transparent explanations, and user-centered interface design. These features help prevent over-reliance, reduce misinterpretation risk, and ensure that the system functions as a supportive decision aid rather than an authoritative admissions oracle.

The results show that even simple models, when embedded within a carefully managed lifecycle, can provide meaningful insights to users while maintaining transparency and interpretability. Likewise, the TF–IDF-based recommendation module demonstrates that classical information retrieval techniques remain effective and explainable alternatives to more complex neural systems, particularly in low-resource settings. The monitoring framework showcases a realistic foundation for long-term maintenance, providing visibility into usage patterns, latency behavior, and user feedback trends that are essential for diagnosing issues and planning future improvements.

As AI-powered advising tools continue to emerge in education, the lessons from *Admit-Guide* underscore the need for systems that balance predictive power with clarity, fairness, and operational discipline. While the prototype does not claim to model real admissions processes, it illustrates the design practices, architectural patterns, and governance considerations that would be required for deploying such a tool responsibly in the real world. Ultimately, this project contributes not only a functional application but also a structured demonstration of how responsible AI principles can be meaningfully implemented across the entire lifecycle of an intelligent system.

Appendix

Appendix A
Example User Journey

A typical user interaction with Admit-Guide proceeds as follows:

1) The user opens the web interface and navigates to the *Admission Predictor* tab.
2) They enter their GRE, TOEFL/IELTS, and CGPA values, along with the name of a target university.
3) The system computes a requirement match score and a model-based difficulty score and returns an estimated admission probability with explanations.
4) The user then switches to the *Program & University Finder* tab, describes their interests in a free-text field, and receives a ranked list of matching programs.
5) Finally, the user may leave feedback in the provided text box, which is logged for future analysis and improvement of the system.

Appendix B
Implementation Notes

The implementation follows standard Python packaging practices. The core pipelines are defined in reusable modules, and configuration (e.g., file paths, port numbers) is isolated into a small configuration file. This separation of code and configuration supports portability across environments (local development, Docker, and potential cloud deployment). All experiments were run on a standard CPU-based laptop, demonstrating that the approach is computationally lightweight.

References

[1] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997.

[2] G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” *Information Processing & Management*, vol. 24, no. 5, pp. 513–523, 1988.

[3] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[4] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[5] M. T. Ribeiro, S. Singh, and C. Guestrin, “Why should I trust you?: Explaining the predictions of any classifier,” in *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2016, pp. 1135–1144.

[6] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[7] R. Bird *et al.*, “Fairlearn: A toolkit for assessing and improving fairness in AI,” *Microsoft Research Technical Report*, 2020.

[8] R. K. Bellamy *et al.*, “AI Fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias,” *IBM Journal of Research and Development*, vol. 63, no. 4/5, pp. 4:1–4:15, 2019.

[9] J. Turnbull, *The Prometheus Monitoring System*. Turnbull Press, 2018.

[10] Grafana Labs, “Grafana documentation,” 2024. [Online]. Available: https://grafana.com/docs

[11] A. Abid, A. Abid, and J. Y. Zou, “Gradio: Hassle-free sharing and testing of ML models in the browser,” 2019. [Online]. Available: https://gradio.app

[12] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*. O’Reilly Media, 2016.

[13] NIST, “Artificial Intelligence Risk Management Framework (AI RMF 1.0),” National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST AI 600-1, 2023.