SLIIT
Discover Your Future

BSc (Hons) in Information Technology Specializing Data Science

Research Project - IT4010 **MLOPs Report**

Group ID: 24-25J-281

Project Title: **AI-Enhanced E-Learning Platform for the Hearing Impaired Children**

1. IT21311840 : Interactive ASL learning sessions using AR and Machine Learning
2. IT21159558 : AI-powered Learning Assistant with Sign Language Support & Recommendations with Performance Prediction
3. IT21173622 : Real-Time Translation of Educational Content to captions, sign language and providing summary.

		IT21311840	IT21159558	IT21173622
Data Pipeline	**Data Sources:**	The data pipeline begins with collecting a comprehensive set of publicly available American Sign Language (ASL) datasets. These datasets encompass thousands of labeled images diverse ASL gestures including alphabets, numbers, and commonly used words. To convert visual data into meaningful numeric features, Google's MediaPipe Hands framework is employed. MediaPipe detects 21 distinct hand landmarks per hand, each with three-dimensional coordinates (x,	• **Chatbot Interaction Dataset:** This custom dataset contains approximately 2000 question-answer pairs, meticulously labeled by subject—Mathematics or English—and by difficulty level: Beginner, Intermediate, and Advanced. The dataset was developed to represent the typical queries and educational needs of hearing-impaired children within the target age group. This labeling empowers the chatbot to operate effectively in two modes: a retrieval-based approach for fast matching of fixed Q&A pairs, and a supervised machine learning classification to infer the subject and	• **Text Dataset for Sentiment Analysis:** This dataset consists of labeled textual data containing sentences and paragraphs annotated with sentiment categories such as positive, negative, and neutral. It serves as the training and validation set for the LSTM-based sentiment analysis model, enabling the system to accurately classify the emotional tone of transcribed captions.

		y, z). These landmarks capture detailed spatial positioning of finger joints, fingertips, and palm center, providing a structured representation of hand pose robust to variations in orientation, scale, and lighting conditions. This landmark-based approach reduces complexity compared to raw image inputs, improving downstream classification performance and model interpretability.	proficiency level, allowing for more nuanced, personalized responses. • **Course Metadata Dataset:** An internally curated dataset catalogs available courses, including detailed metadata such as course titles, descriptive keywords, subject categorization, and difficulty level. This rich dataset supports fine-grained filtering and tailored course recommendations that align with each user's demonstrated skills and interests. Manual curation ensures courses meet educational standards and address the unique requirements of hearing-impaired learners. • **Real-Time Quiz Performance Data (via Firebase):** Quiz responses—including answer correctness, timestamps, and time taken per question—are captured in real time using Firebase's Realtime Database. Firebase's low-latency, synchronized data storage enables the	• **Audio Dataset for Emotion Analysis:** A collection of audio recordings with corresponding emotional labels forms the basis for training audio emotion detection models, including YAMNet and custom CNN/ANN architectures. The dataset encompasses diverse vocal expressions representing emotions such as happiness, fear, sadness, and anger, ensuring robust emotion recognition from speech. • **Text Dataset for Summarization:** This dataset comprises longer textual documents paired with concise, human-generated summaries. It is used to train and fine-tune transformer-based summarization models,

			system to maintain an accurate, up-to-date profile of each student's knowledge and engagement, which is critical for delivering immediate adaptive feedback and powering live predictions of user improvement trajectories. Additionally, Firebase provides robust security through user authentication and data access control. • **Synthetic Game Engagement Dataset:** In early development stages, to compensate for limited real-world data, a synthetic dataset simulating user engagement was programmatically generated (as detailed in the accompanying notebook). This dataset models session-level metrics such as success counts (correct answers), total attempts, engagement time, game difficulty level, and computes a composite "improvement score" derived from a weighted combination	facilitating the generation of brief yet informative summaries of educational video transcripts

			of these factors. This synthesized data forms the foundation for training supervised regression models that predict user learning progress.	
	Data Preprocessing:	Preprocessing is a multi-step process designed to ensure data quality, consistency, and suitability for machine learning training: **1. Data Cleaning:** Initial inspection removes duplicate samples, corrupted files, and entries missing landmark information to avoid noise and bias in the training data. **2. Outlier Detection:** The Isolation Forest algorithm is applied to detect anomalous landmark configurations that could arise from detection errors or extreme hand poses. This unsupervised method isolates rare samples in the high-dimensional landmark space, filtering them out to	• **Chatbot Data Processing:** Raw user queries are subjected to comprehensive cleaning processes including removal of extraneous characters, normalization to lowercase, and elimination of irrelevant punctuation. Tokenization splits text into meaningful units (words or subwords). Stopword removal eliminates common words with little semantic value, improving signal clarity. Lemmatization reduces words to their base or root forms to unify variant word forms. Finally, TF-IDF vectorization transforms cleaned text into sparse, high-dimensional vectors representing the importance of terms relative to the corpus, supporting both efficient	In this project, data preprocessing was conducted separately for audio and text data to prepare inputs suitable for the respective machine learning models, ensuring consistent, clean, and synchronized data across all modalities. **Audio Data Preprocessing** • Audio streams were extracted from educational videos using FFmpeg. • The extracted audio was resampled to a standardized 16 kHz sampling rate to meet the input requirements of the speech recognition (Whisper)

		enhance model generalization and prevent overfitting. **3. Normalization and Scaling:** Landmark coordinates, originally expressed as pixel or relative values, are normalized and scaled to a fixed range (e.g., 0 to 1). This standardization ensures features contribute evenly during training, preventing dominance by larger coordinate values. **4. Column Renaming:** To improve data clarity, columns are renamed systematically (e.g., x0, y0, z0 for landmark 0, and so forth), facilitating consistent access and analysis. **5. Label Encoding:** Categorical gesture labels are converted into integer-encoded classes using label encoding, enabling compatibility with classification algorithms that require numeric inputs. **6. Mean Landmark**	retrieval and input for supervised classifiers. • **Quiz and Engagement Data Processing:** Real-time quiz data streamed from Firebase is normalized and scaled using Min-Max scaling to ensure feature values lie within consistent numeric ranges. Feature engineering creates additional metrics such as average response latency, accuracy trends over recent quizzes, and counts of user sessions, enriching the data for predictive modeling. Statistical outlier detection identifies anomalously fast or slow responses, which are handled appropriately (e.g., exclusion or capping) to maintain data quality and model reliability. • **Course Metadata Vectorization:** Textual elements within course metadata—titles, descriptions, and keywords—are transformed via TF-IDF or dense word embedding	and emotion detection (YAMNet and custom CNN/ANN) models. • Basic audio normalization and noise reduction techniques were applied to enhance clarity and reduce background noise, improving model accuracy. • For emotion detection models requiring raw waveforms, audio data was preserved in waveform format without further transformation. **Text Data Preprocessing for Sentiment Analysis** • Captions generated by the speech-to-text model were cleaned by converting text to lowercase and removing punctuation and special characters to reduce noise.

		Computation: For enhanced user feedback, mean landmark positions per finger and per gesture class are computed and saved separately, providing a reference for real-time finger position comparison during inference. This preprocessing pipeline transforms raw landmark data into a clean, normalized, and well-labeled dataset optimized for high accuracy and robust classification.	techniques (e.g., GloVe, FastText) into numerical vectors that capture semantic relationships. Categorical variables like subject and difficulty are encoded (e.g., one-hot encoding) to facilitate filtering and similarity calculations in recommendation algorithms. • **Synthetic Dataset Preparation:** Input features in the synthetic dataset (success rates, attempt counts, engagement time, level) are uniformly scaled to maintain consistent ranges across variables. The improvement score target is computed by combining normalized features with domain-informed weights, reflecting educational priorities such as mastery and engagement. This preparation ensures that machine learning models can learn meaningful patterns correlating engagement behaviors with measurable improvement.	• The cleaned text was tokenized into sequences of words. • To maintain uniform input size for the LSTM model, sequences were padded or truncated to a fixed length. • This preprocessing ensured that the textual input was formatted consistently for sentiment classification. **Text Data Preprocessing for Summarization** • Full transcripts were tokenized and formatted to comply with the input size constraints of the transformer-based summarization model. • Long transcripts exceeding model input length limits were

				split into smaller chunks for processing. • The summaries generated from these chunks were concatenated to form a comprehensive summary of the video content. **Synchronization** • Throughout the preprocessing pipeline, timestamps were carefully preserved and managed to maintain synchronization between the audio, textual captions, sign language GIF displays, and sentiment/emotion outputs. • This synchronization is critical to ensure a seamless and accurate multimodal learning experience during video playback

	Data storage and versioning	Processed datasets and mean finger landmarks stored in CSV format. GitHub used for version control ensuring traceability and reproducibility.	• **Firebase Integration:** Firebase Realtime Database provides a robust, secure, and low-latency cloud solution for storing live quiz performance and user interaction data. It enables seamless synchronization of data between user devices and backend services, ensuring that the system maintains real-time awareness of student progress for immediate adaptive feedback and prediction. Firebase's integrated user authentication and fine-grained security rules protect user privacy and prevent unauthorized access. • **Version Control of Datasets and Models:** All key datasets—including chatbot Q&A pairs, course metadata, and synthetic datasets—are maintained within GitHub repositories under strict version control. This setup enables detailed tracking of dataset evolution, collaborative	**Data Storage:** All raw and processed data—including educational videos, extracted audio, transcripts, and labeled datasets for sentiment and emotion analysis—are securely stored on cloud-based platforms and dedicated servers. This setup guarantees data availability, backup, and efficient access during model training and inference. **Model Artifacts:** Trained models, checkpoints, and configuration files are systematically stored with proper metadata to facilitate deployment, evaluation, and future retraining. **Version Control:** The project's source code, including preprocessing scripts, model architectures, training routines, and API code, is maintained in Git repositories hosted on platforms such as GitHub. This	

			improvements, and rollback capability, ensuring experimental reproducibility and data integrity. • **Model Artifact Management:** Trained machine learning models such as the Naive Bayes classifier, Random Forest regressor, and ARIMA forecasting models are serialized using Joblib. These artifacts are stored with explicit version tags within GitHub or secure cloud storage environments to facilitate deployment, version tracking, and comparative evaluation. • **In-Memory Session Data:** To optimize performance, transient user session data is maintained in-memory during API calls to allow rapid processing and response generation. Where needed, session data can be persisted into Firebase for long-term storage, enabling historical	enables collaborative development, change tracking, and rollback capability. **Data and Model Versioning:** Large datasets and model files are versioned using tools like Data Version Control (DVC) or Git Large File Storage (Git LFS), ensuring that every change in data or model weights is tracked and reproducible. Model versions are tagged with metadata such as training parameters and performance metrics to aid in selection and deployment.

			analyses and model retraining with expanded datasets.	
Model Development	**Model Selection**	The process of selecting the most appropriate machine learning model for American Sign Language (ASL) gesture recognition was carried out with thorough experimentation and evaluation. Given that the input data consists of tabular features derived from 3D hand landmarks extracted using MediaPipe, the following supervised learning algorithms were considered due to their proven effectiveness on structured data:	A carefully balanced approach was undertaken during model selection to meet the project's needs for accuracy, interpretability, computational efficiency, and scalability: • **Chatbot Models:** ○ ***Retrieval-Based Approach:*** Employing TF-IDF vectorization combined with cosine similarity provides a straightforward, interpretable method for mapping user queries to the closest known questions. This model requires no training and delivers immediate, reliable answers for fixed question sets, making it ideal for frequently encountered educational queries. ○ ***Supervised Classification:*** The Multinomial Naive Bayes	• **Speech Recognition:** The OpenAI Whisper model was chosen for converting audio from educational videos into text transcripts. Whisper's transformer-based architecture excels in handling varied audio conditions and supports multilingual transcription, making it ideal for real-time caption generation. • **Sentiment Analysis:** A Long Short-Term Memory (LSTM) network was employed to classify the sentiment of the transcribed text. LSTM models are well-suited for sequential data and capture contextual dependencies in language, improving sentiment prediction accuracy compared to traditional classifiers.

		• **Random Forest Classifier (RFC):** An ensemble learning technique that builds numerous decision trees during training time and outputs the class that is the mode of the classes output by individual trees. Random Forests are robust to overfitting, handle high-dimensional data well, and provide interpretability through feature importance scores. • **Gradient Boosting Classifier:** Builds trees sequentially, where each subsequent tree attempts to correct the errors of the previous one. This boosting approach often results in high accuracy but can be prone to	classifier was chosen for text classification due to its simplicity, computational efficiency, and proven robustness when working with high-dimensional sparse data like TF-IDF vectors. It effectively captures probabilistic independence assumptions and supports nuanced classification of user questions by subject and difficulty, enabling personalized response generation. • **Recommendation System:** A hybrid recommendation framework blends: ○ ***Content-Based Filtering:*** Leveraging cosine similarity between normalized user quiz performance vectors and TF-IDF vectorized course metadata to ensure recommended courses align closely with demonstrated learner skills and interests.	• **Emotion Detection:** The project incorporated both pre-trained and custom models for audio emotion detection. YAMNet, a pre-trained deep neural network trained on the AudioSet dataset, was used for initial emotion classification from raw audio. To enhance performance, custom Convolutional Neural Networks (CNN) and Artificial Neural Networks (ANN) were developed and trained on labeled emotional speech datasets to detect specific emotions such as happiness, sadness, anger, and fear with greater precision. • **Text Summarization:** Transformer-based models, including BART and T5, were implemented for generating abstractive summaries of long transcripts. These models utilize attention mechanisms to effectively

		overfitting if not properly regularized. • **XGBoost (Extreme Gradient Boosting):** An optimized implementation of gradient boosting designed for speed and performance. XGBoost supports regularization and parallelization, making it well suited for large datasets. • **CatBoost:** A gradient boosting algorithm specifically designed to handle categorical features efficiently and reduce prediction shift with ordered boosting. CatBoost uses symmetric trees and employs techniques to reduce overfitting and improve generalization.	○ ***Planned Collaborative Filtering:*** Intended to analyze patterns in user behaviors and peer preferences to further enhance recommendation diversity and relevance as more user data becomes available. • **Performance Prediction Models:** ○ The Random Forest Regressor was selected for its ability to model complex nonlinear interactions between engagement features and improvement scores, its robustness to outliers and noise, and the capacity to provide interpretable feature importances—critical for understanding which factors most influence learning progress. ○ Linear Regression serves as a complementary baseline, valued for its transparency and	capture the semantic structure of the text, enabling the creation of concise and coherent summaries.

		Model Selection Strategy: Each model was trained and evaluated on the preprocessed landmark dataset using the same stratified train-test split. Key performance metrics such as accuracy, precision, recall, F1-score, and training/inference time were recorded to comprehensively compare models. Due to its ability to handle categorical data natively, faster training time, and slightly higher accuracy, **CatBoost** was selected as the final model. It achieved an overall accuracy of approximately **99.51%**, outperforming the other candidates, which hovered around 99% accuracy.	straightforward interpretability, enabling comparisons to assess the value added by more complex models. • **Time-Series Forecasting:** The ARIMA (AutoRegressive Integrated Moving Average) model was selected for time series forecasting of user engagement trends and future improvement predictions. Its statistical rigor, capacity to model seasonality and autocorrelations, and interpretability make it well-suited for forecasting future learning trajectories and supporting timely pedagogical interventions.	

	Model Training	Training the chosen CatBoost model was approached with careful attention to balancing accuracy, generalization, and computational efficiency: 1. **Data Splitting:** The dataset was split into training and testing sets using **stratified sampling** to maintain consistent class distribution across sets, which is crucial given the multi-class nature of ASL gestures. 2. **Baseline Training:** Initial training runs were performed using default CatBoost parameters to establish baseline performance metrics. This step helped identify the model's natural	Training procedures were meticulously designed to maximize generalization while avoiding overfitting: • **Chatbot Model Training:** ○ The dataset was divided using stratified sampling to maintain balanced class proportions across subject and difficulty labels in both training and testing subsets. ○ The TF-IDF vectorizer was fit exclusively on the training data to prevent leakage of information from the test set. ○ Hyperparameter tuning of the Naive Bayes classifier (e.g., smoothing parameter alpha) was conducted using grid search to achieve the optimal tradeoff between bias and variance. ○ Performance metrics—including accuracy, precision, recall, and F1-score—were computed per	• **LSTM Sentiment Model:** The LSTM model was trained on a text dataset labeled with sentiment categories. Preprocessing involved tokenizing text, converting tokens into embeddings, and padding sequences to a fixed length. The model architecture consisted of embedding layers followed by one or more LSTM layers and dense output layers. Hyperparameters such as learning rate, batch size, dropout rate, and number of epochs were tuned using validation data to optimize performance and prevent overfitting. The Adam optimizer and cross-entropy loss function were employed during training. • **Custom CNN and ANN for Emotion Detection:** Audio datasets containing emotional vocalizations were used to train the CNN and ANN models. Audio waveforms were processed and converted into features compatible with the network input. The CNN architecture included

		strengths and areas needing tuning. 3. **Hyperparameter Tuning:** To optimize model performance, both **Grid Search** and **Randomized Search** techniques were used to explore a range of hyperparameters, including: ○ **Iterations:** Number of boosting rounds (trees) ○ **Learning rate:** Step size shrinkage used to prevent overfitting ○ **Depth:** Maximum depth of individual trees ○ **L2 regularization:** To reduce overfitting	class to ensure robust classification performance across all subjects and proficiency levels. • **Recommendation Model Tuning:** ○ Vector representations of courses and user performance profiles were computed via TF-IDF embeddings and normalized quiz results. ○ Empirical threshold tuning for cosine similarity optimized the balance between recommendation precision and recall. ○ Plans include incremental retraining as new user data accumulates, with potential adoption of online learning methods.	convolutional layers to extract temporal and spectral features, followed by pooling and fully connected layers. The ANN was designed as a feedforward network with multiple hidden layers. Both models were trained using categorical cross-entropy loss and optimized with Adam. Data augmentation and regularization techniques were applied to improve generalizability. • **Summarization Model Fine-tuning:** The transformer summarization models were fine-tuned on paired datasets of long-form text and human-generated summaries. Input text was tokenized and split into manageable segments when exceeding model input length limits. The fine-tuning process involved adjusting model weights using supervised learning, with evaluation metrics guiding hyperparameter adjustments such as learning rate and beam search parameters to balance summary length and informativeness.

		Due to computational resource constraints, exhaustive grid searches were constrained, with manual fine-tuning guided by validation results to arrive at the final hyperparameter set. 4. **Cross-Validation:** Five-fold cross-validation was employed during tuning to mitigate overfitting and provide a robust estimate of model performance across different data subsets. 5. **Evaluation Metrics:** Model performance was evaluated using: ○ **Accuracy:** Overall proportion of	• **Performance Prediction Training:** ○ The synthetic dataset was scaled using Min-Max normalization, preparing features and targets for model consumption. ○ Random Forest hyperparameters (number of trees, tree depth) were optimized using randomized search combined with cross-validation to ensure generalizability and prevent overfitting. ○ Linear Regression was trained as a baseline; residual plots and diagnostic statistics were reviewed to validate assumptions. ○ Model evaluation employed multiple metrics: ▪ **R^2:** Quantifies proportion of variance explained. ▪ **MAE:** Represents average prediction error magnitude, easily interpretable.	**Model Evaluation** • The sentiment and emotion classification models were evaluated using metrics including accuracy, precision, recall, and F1-score to measure classification performance across different classes. • Summarization models were evaluated using ROUGE (Recall-Oriented Understudy for Gisting Evaluation) metrics, comparing generated summaries to reference summaries to assess overlap in content. • Cross-validation and early stopping were employed to mitigate overfitting and ensure models generalized well to unseen data. • Confusion matrices and loss curves were analyzed during training and testing to identify areas for improvement.

		correctly classified gestures. ◦ **Precision and Recall:** Calculated per class to assess how well each gesture is identified without false positives or misses. ◦ **F1-score:** Harmonic mean of precision and recall, providing a balanced metric. ◦ **Confusion Matrix:** To analyze class-wise misclassifications and understand error patterns. 6. **Training Efficiency:** CatBoost's implementation allowed model training within a reasonable time frame, leveraging CPU	▪ **RMSE:** Penalizes larger errors more, reflecting model precision. • **ARIMA Model Training:** ◦ Time series were subjected to stationarity tests (Augmented Dickey-Fuller) with differencing applied to stabilize mean and variance. ◦ Model order parameters (p, d, q) were chosen based on minimization of Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) to balance model fit and complexity. ◦ The ARIMA model was validated on a hold-out data segment using Mean Absolute Percentage Error (MAPE), assessing forecasting accuracy.	

		parallelism. The inference latency was measured at approximately **70 milliseconds per frame**, supporting real-time gesture recognition needs. **Training Outcome:** The final CatBoost model demonstrated excellent predictive performance on unseen test data, showing minimal signs of overfitting. Its feature importance analysis confirmed meaningful reliance on spatial landmark positions, aligning with domain knowledge. The model's fast inference time makes it suitable for deployment in interactive, real-time ASL learning systems that		

		provide immediate feedback.		
Model Integration	**Tools used**	Python served as the primary programming language for this component, handling data preprocessing, model training, and backend development. Key libraries included MediaPipe for extracting hand landmarks, CatBoost for efficient and accurate gradient boosting classification, Scikit-learn for supporting utilities such as data splitting and evaluation, and FastAPI for building the backend API. Data visualization was facilitated by Matplotlib and Seaborn. CatBoost was chosen for model training due to its	• **Core Programming Language:** Python 3.8+ • **Machine Learning Libraries:** ○ Scikit-learn for text vectorization (TF-IDF), Naive Bayes, Random Forest, and Linear Regression modeling ○ Statsmodels for ARIMA time series modeling. • **Data Processing:** Pandas and NumPy for efficient data manipulation and preprocessing • **Model Persistence:** Joblib for serializing and deserializing trained models • **Web Framework:** FastAPI for creating RESTful API endpoints	• **FastAPI (Backend):** The backend API is developed using FastAPI, which provides asynchronous and efficient endpoints to handle video uploads, audio extraction, speech-to-text transcription, sentiment and emotion inference, sign language GIF synchronization, and summary generation. FastAPI facilitates smooth communication between machine learning models and the frontend client. • **Flutter (Frontend):** The mobile application is built with Flutter, offering a cross-platform, responsive interface that displays video playback synchronized with captions, sign

		superior accuracy and native handling of categorical features, while Scikit-learn was employed for baseline modeling and preprocessing tasks. Although TensorFlow and PyTorch were considered, they were not used in the final model pipeline. The backend infrastructure was developed using FastAPI, enabling asynchronous, high-performance serving of the trained CatBoost model. Landmark data is extracted client-side via Google MediaPipe Hands, significantly reducing data transmission by sending compact 3D coordinate sets instead of full images. The communication between the	for chatbot interaction, recommendations, and performance predictions • **Real-Time Data Storage:** Firebase Realtime Database for storing live quiz and engagement data, enabling immediate synchronization and low-latency responses • **Version Control:** Git and GitHub for managing source code, datasets, and model artifacts with version history • **Development Environment:** Jupyter Notebooks and Visual Studio Code for data exploration, prototyping, and production development • **Server:** Uvicorn ASGI server for running FastAPI applications in production environments	language GIFs, and sentiment/emotion indicators. Flutter's reactive widget system enables dynamic UI updates triggered by real-time data received from the backend. • **ML Frameworks:** TensorFlow and PyTorch models are deployed behind the FastAPI backend, with standardized RESTful API endpoints exposing inference services for speech recognition, sentiment analysis, emotion detection, and summarization. • **Media Processing:** Server-side tools such as FFmpeg and MoviePy manage video and audio processing tasks, including audio extraction and synchronization metadata preparation.

		client app and backend is maintained through WebSocket, ensuring low-latency, continuous, bi-directional data exchange critical for real-time inference and feedback. Upon receiving landmark data, the backend preprocesses inputs and runs predictions using the CatBoost model, delivering immediate classification results to the client. Additionally, the system performs fine-grained finger-level analysis by comparing user-provided landmarks against precomputed mean landmarks using cosine similarity. This analysis allows the backend to send precise "Correct position"		**Integration Workflow** • Educational videos are uploaded through the Flutter frontend to the FastAPI backend. • The backend extracts audio streams and transcribes them into text using the Whisper speech recognition model. • Generated captions are passed to the LSTM-based sentiment analysis model and mapped to corresponding sign language GIFs based on timestamps for synchronized display. • Audio emotion detection is performed concurrently: YAMNet is used for general audio event classification, while a custom **Convolutional Neural Network (CNN)** model was selected as the primary audio

		or “Incorrect position” feedback for each finger, enhancing user learning through targeted guidance. Flutter was utilized for the mobile application to deliver a cross-platform, interactive AR interface with real-time gesture visualization and feedback. Source code and data versioning were managed through Git and GitHub, promoting collaboration and reproducibility. The development environment primarily consisted of Google Colab for prototyping and GPU-accelerated training, complemented by local IDEs and Jupyter notebooks for development and testing.		emotion detection tool due to its superior accuracy and ability to capture temporal and spectral speech features effectively. Although an ANN model was also explored, CNN outperformed it and was chosen for deployment. • The backend synchronizes all data streams—video, captions, sign language visuals, sentiment, and emotion—and streams real-time updates to the Flutter frontend. • Upon completion of video playback, the backend generates a concise summary of the transcript using transformer-based models and sends it to the frontend for display. **Challenges and Solutions**

				• Maintaining low-latency, real-time communication between the Flutter frontend and FastAPI backend was critical. This was achieved through asynchronous REST calls and WebSocket connections for streaming updates. • Ensuring accurate synchronization across modalities was addressed by precise timestamp management and event-driven UI updates on the Flutter side. • Model interoperability was facilitated by standardized data formats (e.g., JSON with timestamps) to seamlessly integrate diverse model outputs into the unified pipeline.

Model Deployment	**Testing Environments**	The ASL gesture recognition model is deployed on GPU-enabled cloud and edge servers optimized for real-time video processing and inference. This deployment ensures low latency and high availability, enabling multiple users to simultaneously access the system through a Flutter-based mobile application. The backend leverages FastAPI to provide asynchronous APIs with WebSocket support, allowing persistent, low-latency communication between client devices and the server. This communication channel streams hand landmark data efficiently and delivers	• **Unit Testing:** Utilizing FastAPI's TestClient to rigorously verify API routes, including input validation, model inference correctness, and response formatting. • **Integration Testing:** Simulated user workflows encompassing chatbot queries, receiving personalized course recommendations, and observing real-time performance feedback to ensure system components work seamlessly together. • **Load Testing:** Synthetic concurrent user requests using tools like Locust to evaluate system scalability, latency, and stability under peak loads. • **User Acceptance Testing (UAT):** Sessions with representative users, including hearing-impaired children and educators, to gather qualitative feedback on chatbot naturalness,	• **Local Development:** Initial testing and debugging were conducted on local machines equipped with GPUs such as NVIDIA RTX 3090 and Tesla V100 to accelerate model training and inference. This environment allowed rapid iteration on model tuning and pipeline integration. • **Cloud Infrastructure:** For scalable testing and deployment, cloud platforms including AWS, Google Cloud Platform (GCP), and Microsoft Azure were utilized. These platforms provided GPU-accelerated instances for heavy computation tasks and facilitated storage, model serving, and API hosting.

		instant classification results alongside finger-level feedback. Model artifacts and code are version-controlled using Git and GitHub, while joblib is used for model serialization, enabling reliable and repeatable deployment. Continuous monitoring systems oversee inference latency, prediction accuracy, and server health to quickly identify and resolve operational issues. The testing environment comprised both offline and live evaluations. Offline testing utilized stratified train-test splits and five-fold cross-validation to validate model generalization and performance metrics such as	recommendation relevance, and clarity of performance insights. • **Model Validation:** Comprehensive cross-validation and testing on hold-out datasets to confirm models generalize well and resist overfitting.	• **Mobile Devices:** The Flutter frontend application was deployed and tested on Android and iOS devices to validate real-time captioning, sign language synchronization, sentiment and emotion visualization, and overall user experience on diverse hardware specifications. **Deployment Architecture** • **Backend Deployment:** The FastAPI backend, hosting the ML models and media processing services, was containerized using Docker for consistent deployment across environments. Containers were orchestrated using Kubernetes for scalability and fault tolerance during production deployment.

		accuracy, precision, recall, and F1-score. Additionally, pilot user testing was conducted with a group of hearing-impaired children using real-time AR-based gesture practice sessions, confirming the model's accuracy (~94.6%) and feedback effectiveness (~92.3%). Hardware for testing and deployment included high-performance GPUs such as NVIDIA Tesla V100 and RTX 3090 in on-premises setups, as well as cloud GPU instances (AWS EC2 P3, Google Cloud TPUs) to ensure scalable processing power. This hybrid infrastructure allowed thorough model validation before production and		• **Model Serving:** Models were served via RESTful APIs with asynchronous inference endpoints to maintain responsiveness. TensorFlow Serving and TorchServe were considered for optimized serving of respective models, enabling efficient GPU utilization and batching where appropriate. • **Continuous Integration/Continuous Deployment (CI/CD):** Automated CI/CD pipelines were implemented using GitHub Actions to streamline testing, building, and deployment of backend services. This ensured rapid delivery of updates and maintained system stability. **Performance Optimization**

		ensured responsive real-time inference in deployment. Future plans include implementing automated CI/CD pipelines to streamline retraining and redeployment, ensuring the model adapts to new data and maintains peak performance as the system scales.		• **Latency Reduction:** Model inference times were optimized by quantizing models and using mixed-precision computations where possible. Batch processing and asynchronous calls reduced bottlenecks in the data pipeline. • **Resource Management:** GPU resource allocation and memory management were monitored and tuned to prevent contention and maximize throughput. • **Edge Considerations:** While primary deployment is cloud-based, efforts are underway to optimize models for edge deployment on mobile devices using techniques like model pruning and TensorFlow Lite conversion.

	Deployment Platform			
	Deployment Method			
Future Enhancements	**Model Improvement:**	To further improve the American Sign Language gesture recognition system, several enhancements are planned: • **Temporal Modeling:** Integrate sequence models such as Long Short-Term Memory (LSTM) networks or Transformer architectures to capture temporal dependencies in continuous sign sequences. This will enable recognition of multi-gesture phrases and sentences rather than isolated signs. • **Multi-Hand Tracking:** Enhance the system's	• **Chatbot Development:** ○ Transition to transformer-based natural language models (e.g., BERT, DistilBERT, GPT) fine-tuned on educational dialog corpora to provide richer contextual understanding and nuanced conversational abilities. ○ Expand and diversify the question-answer dataset to include additional subjects, finer-grained difficulty distinctions, and support for multiple languages or dialects to broaden accessibility. ○ Enable multimodal input processing, integrating text, audio, and real-time sign gesture recognition, allowing natural and	• **Fine-tuning Emotion Detection:** Further fine-tuning the CNN model on domain-specific emotional speech datasets will improve accuracy and robustness in detecting subtle emotional cues present in educational content. • **Advanced Sentiment Analysis:** Exploring transformer-based models such as BERT or RoBERTa for sentiment classification could enhance understanding of complex linguistic nuances in captions, providing more accurate sentiment detection. • **Multimodal Fusion:** Developing algorithms to effectively combine audio-based

		capability to accurately detect and interpret gestures involving both hands simultaneously. This includes improved landmark association and disambiguation in cases of overlapping or occluded hands. • **Occlusion and Robustness Handling:** Develop advanced algorithms to mitigate the effects of partial hand occlusions, varying lighting conditions, and camera angles, ensuring reliable landmark detection in diverse real-world environments. • **Augmented Feedback Visualization:** Expand real-time feedback mechanisms to include confidence scores, dynamic AR overlays,	inclusive interactions for hearing-impaired users. ○ Implement active learning frameworks to iteratively improve model accuracy by incorporating user corrections and new data dynamically. • **Recommendation Engine:** ○ Introduce collaborative filtering techniques leveraging implicit and explicit user feedback to enhance personalization beyond content-based similarities. ○ Develop reinforcement learning approaches to dynamically adapt course recommendations based on real-time engagement and learning outcomes, optimizing for long-term educational success. ○ Integrate knowledge graph representations to model course dependencies, enabling coherent, scaffolded learning pathways rather than isolated course	emotion and text-based sentiment analysis would provide a richer, more holistic interpretation of the emotional context. • **Sign Language Visualization:** Moving beyond GIFs, integrating avatar-based or 3D animated sign language representations could offer a more natural and comprehensive visual language experience. • **Real-time Adaptation:** Implementing adaptive user interfaces that respond dynamically to detected sentiment and emotion—for example, adjusting playback speed or highlighting key content—would personalize the learning experience.

		and more intuitive visual cues to guide users effectively during practice sessions. • **Data Augmentation and Expansion:** Increase dataset diversity through synthetic data generation, augmentation techniques, and inclusion of more demographic variations to improve model generalization and fairness. • **Automated MLOps Pipeline:** Build continuous integration and deployment pipelines to automate data ingestion, model retraining, testing, and deployment. This will facilitate rapid iteration	suggestions. • **Performance Prediction:** ○ Integrate additional multimodal behavioral and biometric features such as facial emotion recognition and audio sentiment analysis to improve prediction accuracy. ○ Explore advanced deep learning time series models such as LSTM, GRU, or Transformer-based architectures to capture complex temporal dependencies in user learning progress. ○ Develop alerting and notification systems that proactively inform educators or guardians of deteriorating or plateauing performance trends for timely intervention. • **Operational and Accessibility Enhancements:** ○ Optimize models for on-device execution to provide offline	• **Model Optimization for Edge Devices:** Optimizing models through pruning, quantization, and conversion to lightweight formats will enable efficient real-time processing on mobile devices without reliance on cloud services. **Feature Expansion** • **Multi-language Support:** Extending speech recognition, sentiment analysis, and sign language translation to multiple languages and dialects would broaden accessibility. • **User Profile Integration:** Incorporating user profiles and learning history could enable personalized content

		and adaptation to new data, maintaining system performance over time. • **Model Compression and Optimization:** Explore model pruning, quantization, and edge deployment strategies to enable the system to run efficiently on low-resource devices like smartphones without sacrificing accuracy.	chatbot and recommendation functionalities critical for users with limited connectivity. ○ Extend support for multiple sign languages and spoken languages to serve a diverse global learner population. ○ Establish continuous monitoring frameworks for model performance and fairness, with automated drift detection triggering retraining cycles to maintain accuracy and equity.	recommendations and adaptive learning paths. • **Feedback and Analytics:** Adding analytics dashboards for educators to monitor engagement and emotional response trends could inform instructional strategies.