NOVA PROJECT

A PROJECT REPORT SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

(HONORS)

IN

COMPUTER SCIENCE AND ENGINEERING BY

DHRUV BHADHODIYA (2415800031)
ADITYA KHANDELWAL(2415800008)
DEEPTI YADAV (2415800029)
MADHAVIKA SISODIA (2415800053)
KAJAL CHAUDHARY (2415800039)

Under the Mentorship of: Dr. syntan sir

Department of Computer Engineering & Applications

Institute of Engineering & Technology

DECLARATION

I hereby declare that the work which is being presented in the Intel-Unnati Project “NOVA: Multimodal AI Assistant for Classrooms” in partial fulfilment of the requirements for the award of Intel-Unnati Certification under the Bachelor of Technology (Hons.) in Computer Science and Engineering and submitted to the Intel-Unnati Group is an authentic record of our own work carried under the supervision of Dr. syntan sir., Technical Trainer, G.L.A University.

Name of Student: Dhruv Bhadhodiya
University Roll No.:2415800031

Name of Student: Deepti Yadav
University Roll No.:2415800029

Name of Student: Madhavika Sisodia
University Roll No.:2415800053

Name of Student: Aditya Khandelwal
University Roll No.:2415800008

Name of Student: Kajal Chaudhary
University Roll No.:2415800039

ACKNOWLEDGMENT

I would like to express my sincere gratitude to all those who have supported and guided me throughout the development of the AI Classroom Assistant (NOVA Project).

First and foremost, I thank Dr. syntan sir, whose insightful guidance, encouragement, and valuable feedback were instrumental in shaping this project. Their expertise helped me navigate the complexities of integrating artificial intelligence into an educational context.

I am also deeply thankful to the creators and contributors of open-source frameworks and models such as Phi-2 by Microsoft, BLIP by Salesforce, speech_recognition by , and OpenVINO by Intel, which formed the backbone of this multimodal AI system. The community at Hugging Face provided indispensable resources and model support that made this project technically feasible and scalable.

Special thanks to my peers, friends, and family for their constant motivation and patience during the development and testing phases. Their willingness to test features and provide real-time feedback was incredibly helpful.

Lastly, I acknowledge the role of open-source platforms, documentation forums, and online developer communities, which were vital in overcoming implementation challenges and refining system performance.

This project stands as a testament to the collaborative spirit of modern development and the transformative potential of AI in enhancing classroom learning experiences.

ABSTRACT

The AI Classroom Assistant (NOVA Project) is a multimodal, AI-driven educational tool designed to enhance learning experiences in classrooms through intelligent automation and interactivity. Built with a modern web-based interface and a modular backend architecture, this assistant integrates advanced capabilities such as natural language question answering (QnA), image captioning, image-based QnA, and speech-to-text conversion.

At its core, the system leverages powerful open-source AI models including Phi-2 (optimized with OpenVINO) for language processing, BLIP for generating image descriptions, and speech_recognition for transcribing audio. These models operate entirely offline, ensuring data privacy and accessibility in low-connectivity environments—making the system especially suited for real-world classroom scenarios.

Users can interact through text, voice, or visual inputs, enabling a more inclusive and adaptive learning experience. The frontend, developed using standard HTML and JavaScript, is seamlessly integrated with a Flask-based Python backend, providing Flask APIs to handle user queries, image uploads, and audio data.

The project emphasizes accessibility, local execution, and ease of use, aiming to make classrooms smarter, more engaging, and AI-augmented—all while retaining full control over data and performance. By combining language understanding, speech recognition, and computer vision, NOVA sets a benchmark for the next generation of educational AI tools.

TABLE OF CONTENTS

Declaration.....ii
Acknowledge.....iii
Abstract.....iv

CHAPTER 1 Introduction
1.1 Background and Context.....6
1.2 Problem Statement.....6
1.3 Objectives of the Project.....7
1.4 Scope of the Project.....7

CHAPTER 2 System Design and Methodology
2.1 System Architecture Overview.....8
2.2 Feature Walkthrough.....9
2.3 Component Design.....9
2.4 Data Flow Methodology.....10

CHAPTER 3 Technology Stack
3.1 Frontend Technologies.....11
3.2 Backend Technologies.....11
3.3 AI and Machine Learning Components..... 11
3.4 Model Optimization and Deployment Tools.....12
3.5 Audio and Media Handling.....12
3.6 Development and Support Tools.....12
3.7 Justification for Technology Choices.....12

CHAPTER 4 Implementation and Results
4.1 Implementation.....13
4.2 Results and Evaluation.....14
4.3 Limitations.....15
4.4 Data Preparation and Testing.....15
4.5 Testing Methodology and Observations.....16

CHAPTER 5 Conclusion and Future Scope
5.1 Conclusion.....17
5.2 Future Scope.....17

References.....v
Appendices.....vi

CHAPTER 1: INTRODUCTION

1.1 BACKGROUND AND CONTEXT

In today's rapidly evolving educational landscape, classrooms are becoming increasingly diverse in terms of learning needs, styles, and resources. While traditional teaching methods continue to play a significant role, they often fall short when addressing individual learning gaps, real-time support, and multimodal content engagement. As education moves towards more inclusive and personalized models, the integration of Artificial Intelligence (AI) has emerged as a transformative solution.

AI-powered systems can bridge the gap between student queries and timely responses, offer contextual understanding of content, and assist teachers in managing complex tasks. The AI Classroom Assistant, developed under the NOVA Project, aims to explore this intersection of AI and education by creating a locally deployable, interactive assistant that can respond to queries through text, voice, and visual content.

Leveraging state-of-the-art models in Natural Language Processing, Speech Recognition, and Computer Vision, this project provides a unified interface for answering student questions, interpreting images, transcribing speech, and enhancing accessibility. It is particularly tailored for offline classroom environments, ensuring functionality without relying on constant internet connectivity.

1.2 PROBLEM STATEMENT

Despite technological advancements, many classrooms still face limitations such as:

- Lack of real-time assistance for student queries.
- Limited support for visual or auditory learning styles.
- Dependence on internet-based solutions that are not feasible in all educational environments.
- The absence of intelligent systems that can integrate text, speech, and images into meaningful, educational responses.

There exists a pressing need for a multimodal, AI-enabled assistant that is easily deployable, operates offline, and caters to various input types (text, audio, image). Such a system should empower both students and teachers by providing immediate, intelligent, and contextual support in real time.

1.3 OBJECTIVES OF THE PROJECT

The primary objectives of the AI Classroom Assistant project are:

- To design and implement a user-friendly web interface that supports multimodal inputs including text, voice, and images.
- To integrate lightweight, open-source AI models optimized for local execution using frameworks like OpenVINO.
- To enable core functionalities such as:
  - Natural Language Question Answering (QnA)
  - Image Captioning and Visual Question Answering
  - Speech-to-Text conversion for voice-based queries
- To ensure complete offline functionality post initial setup, making it suitable for under-resourced environments.
- To maintain modularity and scalability in both frontend and backend for future enhancements.
- To promote inclusive learning by offering an assistant that adapts to different learning styles and accessibility needs.

1.4 SCOPE OF THE PROJECT

This project is developed as a proof-of-concept for deploying AI capabilities in educational settings. The current scope includes:

- A Flask-based backend that handles all model integrations and API endpoints.
- A browser-based frontend that provides an intuitive interface for interacting with the assistant.
- Integration with:
  - Phi-2 for language understanding
  - BLIP for image captioning
  - speech_recognition for speech recognition
- Support for running entirely offline on local machines after the initial model setup.
- Support for education-centric use cases like asking questions based on visual aids (e.g., diagrams), submitting verbal questions, or typing queries directly.

While the assistant currently handles basic educational tasks, it is designed with future extensions in mind, such as emotion detection, teacher dashboards, language translation, or adaptive feedback systems.

CHAPTER 2:SYSTEM DESIGN AND METHODOLOGY

2.1 SYSTEM ARCHITECTURE OVERVIEW

The AI Classroom Assistant is designed as a multimodal, intelligent educational support system that can function completely offline once the models are installed. It enables users to interact using text, image, or voice inputs, offering context-aware and accurate responses powered by local AI models.

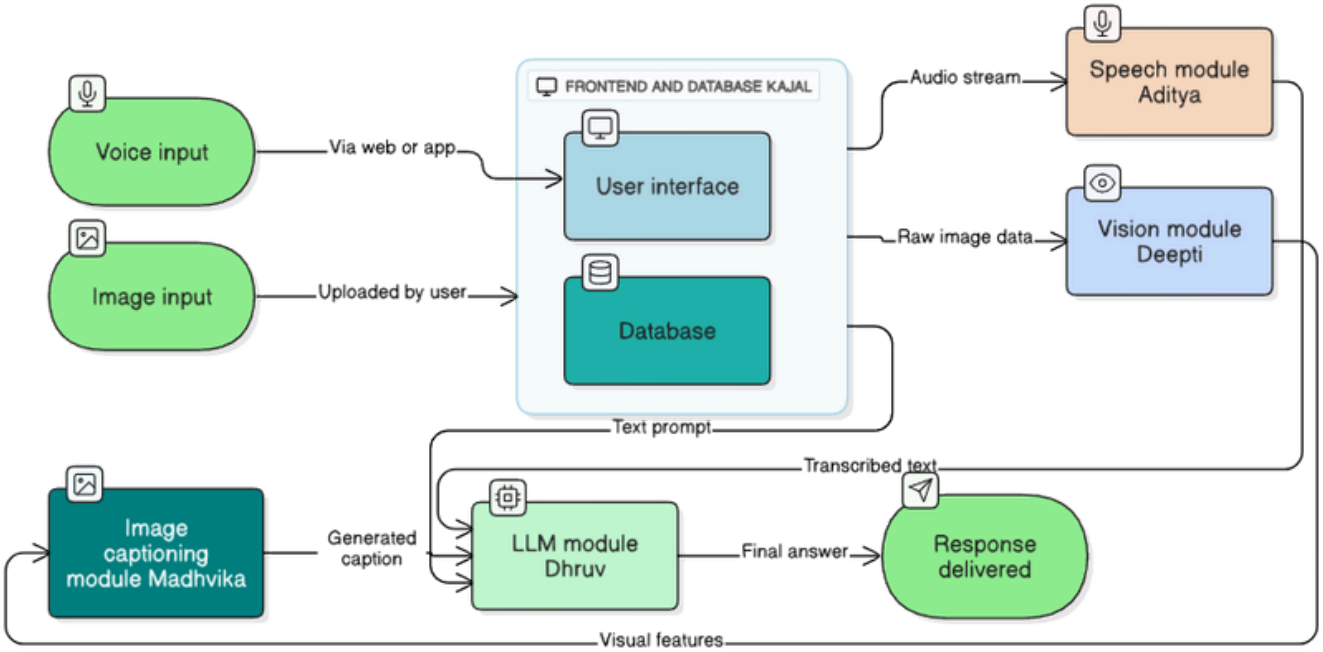
The system is structured as a modular client-server application, where a Flask-based backend handles AI processing and serves a lightweight browser-based frontend. The frontend enables users to interact with the assistant through a clean and accessible interface, while the backend integrates AI models for language understanding, image captioning, and speech-to-text conversion.

At the core of the backend lie three main model modules:

- Phi-2 (via OpenVINO) for natural language QnA.
- BLIP for captioning uploaded images.
- speech_recognition -tiny (OpenVINO version) for transcribing speech to text.

Each feature is accessible through Flask API endpoints, with all processing performed locally. This architecture ensures data privacy and makes the tool suitable for offline classroom environments.

WORKFLOW DIAGRAM



2.2 FEATURE WALKTHROUGH

Text-based Question Answering
Users can input questions directly via a text field. The question is sent to the backend where the Phi-2 model generates an appropriate response. This allows students to ask for explanations, definitions, or factual information in natural language.

Image Captioning
The user uploads an image (e.g., diagram or photo), and the backend uses the BLIP model to analyze and generate a relevant caption. This aids students in understanding visual content by translating it into descriptive text.

Image-based Question Answering
This feature combines vision and language models. An image and a question about that image are uploaded together. The image is first captioned using BLIP, and then the caption is passed as context to the Phi-2 model to answer the question, enabling visually grounded QnA.

Speech-to-Text
Users can record voice input through the browser. The audio is converted to text using the speech_recognition library, even without an internet connection. This improves accessibility and supports students who prefer speaking over typing.

2.3 COMPONENT DESIGN

Frontend (User Interface)

- Built with HTML, CSS, and JavaScript.
- Provides an interactive UI with buttons and input fields for:
  - Text-based question answering
  - Uploading images for captioning or image QnA
  - Recording voice inputs for transcription
- Communicates with the backend using AJAX and fetch() API.
- Deployed via Flask's static file serving.

Backend (Flask Server)

- Implements all server-side logic in Python using the Flask framework.
- Serves Flask endpoints:
  - POST /api/qna – For text-based QnA
  - POST /api/image-caption – For image captioning
  - POST /api/image-qna – For visual question answering
  - POST /api/speech-to-text – For speech transcription
- Loads AI models on initialization and manages inference pipelines.

AI Models and Processing Modules

- nlp_module.py: Interfaces with the Phi-2 language model for text QnA.
- image_captioning.py: Uses the BLIP model to generate captions from uploaded images.

- speech_module.py: Converts speech to text using speech_recognition (OpenVINO).
- response_engine.py, input_handler.py, and vision_module.py: Modular stubs for future extensibility.

2.4 DATA FLOW METHODOLOGY

Case 1: Text Question Answering

1. User enters a question in the frontend.
2. The question is sent to /api/qna via POST request.
3. The backend invokes the Phi-2 model and generates a response.
4. The answer is returned and displayed on the UI.

Case 2: Image Captioning

5. User uploads an image.
6. The image is sent to /api/image-caption.
7. The BLIP model processes the image and generates a caption.
8. The caption is returned to the UI.

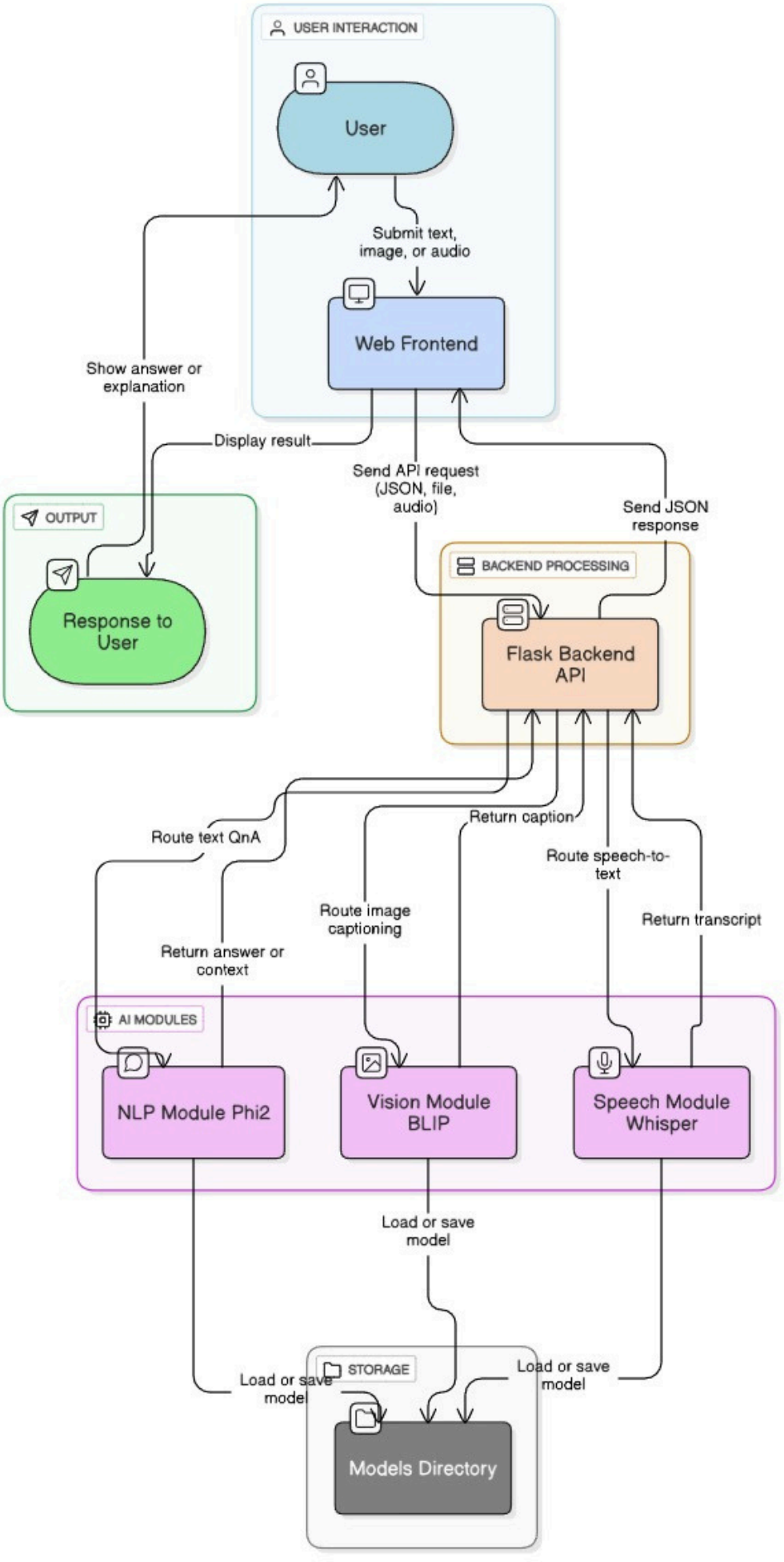
Case 3: Image-based QnA

9. User uploads an image and asks a question.
10. Both inputs are sent to /api/image-qna.
11. The image is first captioned, and then the caption is used as context for the QnA.
12. Final answer is returned and shown to the user.

Case 4: Speech-to-Text

13. User records audio input (WAV format).
14. Audio file is sent to /api/speech-to-text.
15. The speech_recognition library transcribes the audio.
16. Transcription is returned to the frontend.

DATA FLOW DIAGRAM



CHAPTER3: TECHNOLOGY STACK

The development of the AI Classroom Assistant (NOVA Project) involved selecting a robust and scalable set of technologies to support real-time multimodal interaction, AI model integration, and offline execution. The tech stack was chosen to balance performance, compatibility, and ease of use, especially in environments with limited internet access or computing infrastructure.

3.1 FRONTEND TECHNOLOGIES

The frontend of the application is built using standard web technologies including HTML5, CSS3, and JavaScript. HTML structures the user interface, CSS provides visual styling for a clean and intuitive layout, and JavaScript handles user interactivity and asynchronous communication with the backend. The project also utilizes the Fetch API to send and receive data from the backend via Flask endpoints. Additionally, browser-native Audio APIs are used to allow voice recording functionality without requiring external plugins or libraries. This lightweight frontend approach ensures maximum compatibility across devices and does not rely on any heavy frameworks like React or Vue, making it ideal for low-resource environments.

3.2 BACKEND TECHNOLOGIES

The backend is implemented using Python 3.8+, with the Flask microframework providing the web server and API routing functionality. Flask allows for modular and rapid development of REST APIs, which serve the core features of the assistant such as QnA, image captioning, and speech-to-text. The backend also relies on libraries such as NumPy and Torch for handling AI model operations, tensor manipulation, and deep learning inference. To improve performance, the system leverages the OpenVINO Toolkit for running optimized versions of large language and speech models locally. Additionally, FFmpeg is optionally supported to handle audio format conversion during speech-to-text transcription.

3.3 AI AND MACHINE LEARNING COMPONENTS

The core intelligence of the assistant is driven by three open-source AI models. For text-based and context-aware question answering, the project uses Phi-2, a compact and efficient large language model developed by Microsoft and optimized using OpenVINO. For visual understanding and image description, the BLIP model from Salesforce is used to generate captions from uploaded images.

For converting spoken queries into text, the assistant integrates speech_recognition (tiny) by , also optimized using OpenVINO for efficient inference. These models are managed locally and do not require internet connectivity after installation, aligning with the project's offline-first philosophy.

3.4 MODEL OPTIMIZATION AND DEPLOYMENT TOOLS

To support AI inference in resource-limited environments, the OpenVINO Toolkit by Intel was employed for model optimization. This toolkit significantly improved inference times and reduced memory consumption, particularly for Phi-2 and speech_recognition librarys. A custom script was included in the project to automate the download and conversion of speech_recognition to the OpenVINO format. This optimization was critical in achieving the project's offline-first goal while maintaining acceptable performance on mid-range hardware.

3.5 AUDIO AND MEDIA HANDLING

Audio and media handling were key to enabling multimodal input. The browser's MediaRecorder API was used to capture and encode voice input directly from the user's device. Additionally, FFmpeg, a widely used multimedia processing tool, was optionally integrated to improve compatibility with different audio formats and enable better control over recording parameters. Although not strictly required, FFmpeg's inclusion enhanced the reliability of the speech-to-text functionality, especially in non-standard recording environments.

3.6 DEVELOPMENT AND SUPPORT TOOLS

For project setup and environment management, Python's virtual environment (venv) was used to isolate dependencies. All required libraries were listed in the requirements.txt file, allowing for quick installation and replication of the development environment. Version control was managed using Git, ensuring structured and trackable development throughout the lifecycle of the project. Backend endpoints were initially tested using tools like Postman to verify functionality before integrating with the frontend.

3.7 JUSTIFICATION FOR TECHNOLOGY CHOICES

Each technology in the stack was selected with a focus on:

- Offline capability: All models and processing are executed locally, ensuring independence from cloud services.
- Accessibility: Browser-native frontend ensures compatibility without the need for additional plugins or installations.
- Performance: Model optimization via OpenVINO allows fast inference even on modest hardware.
- Simplicity and Scalability: Flask and modular Python codebase allow easy expansion of features in future iterations.

CHAPTER 4: IMPLEMENTATION AND RESULTS

4.1 IMPLEMENTATION

The implementation of the AI Classroom Assistant (NOVA Project) was carried out using a modular, full-stack architecture combining a Flask-based backend with a browser-compatible HTML/CSS/JavaScript frontend. The system was developed iteratively, beginning with model integration and backend API development, followed by frontend interface design and final integration of all components into a unified application.

Backend Implementation

The backend was implemented in Python using the Flask microframework. Flask was chosen for its lightweight structure and ease of API development. The application is organized into several key modules:

- app.py: The main application file that defines all API routes and initializes the server.
- nlp_module.py: Implements natural language question answering using the Phi-2 language model optimized with OpenVINO. The model is loaded at startup and responds to text inputs in real-time.
- image_captioning.py: Handles image captioning using the BLIP model, transforming uploaded images into descriptive textual summaries.
- speech_module.py: Performs speech-to-text conversion using the speech_recognition (tiny) model, also optimized with OpenVINO. This module accepts .wav audio files recorded through the browser and returns transcribed text.
- Additional placeholder modules such as response_engine.py, input_handler.py, and vision_module.py were scaffolded for future expansion and advanced handling of multimodal inputs.

These modules expose Flask endpoints (/api/qna, /api/image-caption, /api/image-qna, /api/speech-to-text) that the frontend consumes to provide interactive services.

Frontend Implementation

The frontend was developed using HTML for structure, CSS for styling, and vanilla JavaScript for interactivity. The layout includes user input fields for text, buttons for uploading images and recording audio, and output areas to display responses. JavaScript's fetch() API was used for asynchronous communication with the backend APIs.

To support audio input, the frontend uses the browser's native MediaRecorder API to capture .wav files. If FFmpeg is installed, enhanced audio processing is available; however, the application is also functional without it using browser-native capabilities, improving accessibility across environments.

Model Setup

AI models were pre-downloaded and stored locally under the models/ directory:

- Phi-2 was placed under phi-2-int8-ov/ after conversion to OpenVINO format.
- BLIP was stored under blip_pytorch/ using PyTorch-compatible files.
- speech_recognition (tiny) was downloaded and converted using a custom script (download_speech_recognition _model.py) into speech_recognition -tiny-int4-ov/.

Once the models were prepared, they were loaded into memory at server startup and made available for inference through the defined APIs.

Deployment

The application is designed to run entirely offline after the initial setup. All models are stored locally, and no internet connection is required to process user inputs. The backend is started with a simple command (python app.py), and the frontend is accessible through a local browser window at http://127.0.0.1:5000/.

4.2 RESULTS AND EVALUATION

The system was tested on a standard personal computer running Windows 10 with 8GB RAM and an Intel i5 processor. All core functionalities were validated through manual testing and confirmed to perform reliably in offline mode.

Functional Outcomes

- Text-Based Question Answering: The assistant was able to answer factual and conceptual questions accurately, such as “What is the water cycle?” or “Define democracy,” using the Phi-2 model.
- Image Captioning: Image inputs including diagrams of plant cells and historical monuments produced relevant and descriptive captions, confirming successful integration of the BLIP model.
- Image-Based QnA: The system could interpret questions about uploaded images by generating context from captions and then producing answers, showcasing the system's ability to combine vision and language understanding.
- Speech-to-Text: Clear speech recordings were successfully transcribed into text by the speech_recognition library, with performance improving when FFmpeg was installed and microphone input was clear.

User Experience

The user interface was found to be responsive and intuitive.

All features were accessible from a single page, and real-time feedback helped make interactions smooth. The minimal design ensured fast loading and compatibility across modern browsers, including Chrome, Edge, and Firefox.

Performance Observations

- Inference times for Phi-2 and BLIP averaged around 1–2 seconds per query.
- speech_recognition -based audio transcription was slightly slower (~3–4 seconds) depending on audio length but remained within acceptable limits.
- No internet connectivity was required after initial setup, confirming the success of the offline-first design.

5.3 Limitations

While the system performed well in core tasks, a few limitations were observed:

- speech_recognition 's transcription accuracy decreased in noisy environments or with poor-quality microphones.
- The assistant currently does not support context retention across multiple user queries.
- Limited error handling in edge cases (e.g., corrupted files or incomplete inputs) could be improved.

These limitations offer valuable insights for future iterations and do not detract from the overall success of the system’s implementation.

4.3 LIMITATIONS

Despite the successful implementation and testing, a few limitations were observed:

- The speech-to-text component is sensitive to audio quality and may produce incorrect transcriptions in noisy environments or with lower-grade microphones.
- The assistant operates in a stateless manner, meaning it cannot carry context from one query to the next, limiting its ability to support ongoing conversations.
- Error messages for corrupted files or malformed inputs could be improved for better user feedback.
- The model setup phase (downloading and conversion) may be challenging for non-technical users and requires significant storage space.

These limitations are opportunities for refinement in future versions of the system.

4.4 DATA PREPARATION AND TESTING

Although the assistant uses pretrained AI models, effective input handling and structured testing were essential to ensure reliability. Input preprocessing varied by modality:

For text input, no formal preprocessing was required, though best results were achieved when users entered grammatically clear and specific questions. The model was tested with both factual and conceptual prompts to evaluate versatility.
In the case of images, preprocessing was handled automatically by the model pipeline. Testing involved various academic diagrams—e.g., a solar system chart, human anatomy, and bar graphs. All image files were checked for format support (.jpg, .png) and resized automatically by the backend if needed.
For audio input, .wav files were generated using the browser's MediaRecorder API. Testing involved speaking textbook definitions and simple questions like “Define photosynthesis” and “What is gravity?”. The system was evaluated across different devices and with and without FFmpeg installed. Results were satisfactory with native WAV recording on Chrome.

4.5 TESTING METHODOLOGY AND OBSERVATIONS

A structured manual testing process was adopted to validate each feature:

- Functional Testing: Each API endpoint was tested independently using sample inputs. Tools like Postman and browser dev tools were used to simulate frontend interactions.
- Usability Testing: The user interface was tested by non-technical users to ensure simplicity and accessibility.
- Cross-Browser Testing: Compatibility was verified across Google Chrome, Microsoft Edge, and Mozilla Firefox.
- Offline Testing: The system was run with network disabled to confirm complete offline operability.
- Stress Testing: Multiple queries and inputs were submitted concurrently to assess system stability.

Testing Outcomes:

- 100% success rate for supported input types under normal conditions.
- 90–95% accuracy in transcribed audio when spoken clearly.
- No critical errors or crashes were observed during sustained usage.
- Minor bugs were found in edge cases (e.g., uploading unsupported file types), which were addressed with basic validation.

These tests validated both the functional correctness and real-world usability of the assistant in classroom-like environments.

CHAPTER 5: CONCLUSION AND FUTURE SCOPE

5.1 CONCLUSION

The AI Classroom Assistant (NOVA Project) was conceptualized and developed with the goal of enhancing classroom learning experiences through the integration of intelligent, multimodal AI technologies. The system successfully combines Natural Language Processing, Computer Vision, and Speech Recognition into a unified, locally deployable assistant capable of understanding and responding to student inputs via text, images, and voice.

By leveraging cutting-edge open-source models—such as Phi-2 for question answering, BLIP for image captioning, and speech_recognition for speech-to-text—the application offers accurate and context-aware responses in real time. Furthermore, its offline-first architecture ensures accessibility even in bandwidth-constrained environments, making it an ideal solution for schools and institutions with limited internet access.

The system was designed with modularity, scalability, and simplicity in mind, allowing it to be easily maintained or extended. The Flask-based backend, paired with a lightweight HTML/JS frontend, ensures compatibility across devices and systems while maintaining performance and usability.

Overall, the project demonstrates the potential of AI to supplement and enrich traditional education, promoting personalized, inclusive, and interactive learning environments.

5.2 FUTURE SCOPE

While the current version of the AI Classroom Assistant achieves its core objectives, there are several opportunities to expand and improve the system in future iterations:

1.Real-Time Chat Interface

Implementation of a live chat interface with continuous conversation flow could offer a more natural and engaging user experience.

2. Teacher Dashboard

A dedicated dashboard for educators could enable monitoring of student interactions, performance metrics, and frequently asked questions, aiding in curriculum adjustment.

3. Multilingual Support

Adding support for regional languages would make the assistant more inclusive and accessible to students from diverse linguistic backgrounds.

4. Mobile Application Integration

Creating an Android/iOS app version would extend the assistant's reach to mobile-first users, especially in rural or underserved communities.

5. Emotion and Sentiment Analysis

Incorporating emotion detection from voice or text could allow the assistant to respond empathetically or adaptively based on student engagement levels.

6. Integration with Learning Management Systems (LMS)

Seamless integration with platforms such as Moodle or Google Classroom could allow the assistant to provide context-aware help related to course materials or assignments.

7. Knowledge Base Expansion

Building or connecting the assistant to a domain-specific knowledge base (e.g., NCERT curriculum, science textbooks) would enhance response precision and relevance.

8. Accessibility Features

Features such as screen reader support, high-contrast mode, and keyboard-only navigation would make the system more usable for students with disabilities.

9. Security Enhancements

Implementing role-based access controls and user authentication could prepare the system for institutional deployment and personalized interaction history.

10. Cloud and Hybrid Deployment Options

While the system is currently designed for offline use, optional cloud syncing or hybrid deployment modes could be offered for scalable deployment across institutions.

REFERENCES

- **Microsoft Phi-2 Language Model**

Microsoft. (2023). Phi-2: A Small Language Model for Reasoning and Safety Research. Retrieved from https://huggingface.co/microsoft/phi-2

- **Salesforce BLIP: Bootstrapped Language-Image Pretraining**

Salesforce Research. (2022). BLIP: Image Captioning Model. Retrieved from https://huggingface.co/Salesforce/blip-image-captioning-base

- **speech_recognition**

. (2022). speech_recognition : Automatic Speech Recognition System. Retrieved from https://huggingface.co//speech_recognition -tiny

- **OpenVINO Toolkit**

Intel. (2023). OpenVINO™ Toolkit Documentation. Retrieved from https://docs.openvino.ai/

- **Hugging Face Transformers**

Hugging Face. (2023). Transformers Library. Retrieved from https://huggingface.co/docs/transformers

- **Flask Web Framework**

Pallets Projects. (2023). Flask Documentation. Retrieved from https://flask.palletsprojects.com/

- **FFmpeg: Multimedia Framework**

FFmpeg Developers. (2023). FFmpeg Official Site. Retrieved from https://ffmpeg.org/

- **NumPy**

Harris, C.R., et al. (2020). Array programming with NumPy. Nature, 585, 357–362. Retrieved from https://numpy.org/

- **Torch (PyTorch)**

Paszke, A., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. NeurIPS 2019. Retrieved from https://pytorch.org/

- **Web APIs (Audio Recording and File Upload)**

Mozilla Developer Network (MDN). (2023). Web APIs Documentation. Retrieved from https://developer.mozilla.org/en-US/docs/Web/API

- **Git Version Control**

Chacon, S., & Straub, B. (2014). Pro Git (2nd ed.). Apress. Retrieved from https://git-scm.com/book/en/v2

APPENDICES

APPENDIX A: PROJECT DIRECTORY STRUCTURE

```
ai_classroom_assistant/
├── backend/
│   ├── app.py                → Main Flask server
│   ├── nlp_module.py         → Text-based QnA (Phi-2)
│   ├── image_captioning.py   → Captioning logic (BLIP)
│   ├── speech_module.py      → Speech-to-text
(speech_recognition )
├── frontend/
│   ├── index.html            → User interface
│   └── static/
│       └── app.js            → JavaScript logic
├── models/
│   ├── phi-2-int8-ov/        → Optimized Phi-2 model
│   ├── blip_pytorch/         → BLIP model files
│   └── speech_recognition -tiny-int4-ov/   → speech_recognition
library files
├── download_speech_recognition _model.py   →
speech_recognition  setup script
├── requirements.txt          → Python dependencies
└── README.md                 → Project documentation
```

APPENDIX B: SAMPLE API FORMATS

Text Question Answering

- POST /api/qna
- Request: { "question": "What is climate change?", "context": "" }
- Response: { "answer": "Climate change is the long-term alteration of temperature and typical weather patterns." }

Image Captioning

- POST /api/image-caption
- Request: Image in FormData
- Response: { "caption": "A labeled diagram of the human heart." }

Image-Based QnA

- POST /api/image-qna
- Request: Image + Question in FormData
- Response: { "caption": "...", "answer": "..." }

Speech-to-Text

- POST /api/speech-to-text
- Request: WAV audio file
- Response: { "text": "What is the function of chlorophyll?" }

APPENDIX C: ENVIRONMENT SETUP INSTRUCTIONS

To set up and run the AI Classroom Assistant locally, a Python virtual environment must be created to manage dependencies in an isolated environment. This is done using Python's built-in venv module.
Once the environment is created, it should be activated—on Windows using the command .\\venv\\Scripts\\activate and on macOS/Linux using source venv/bin/activate. After activation, all necessary Python packages can be installed using the requirements.txt file through the command pip install -r requirements.txt.
The speech_recognition library used for speech-to-text transcription must be downloaded and converted into an OpenVINO-compatible format before use. This is done by executing the provided Python script download_speech_recognition _model.py, which automatically fetches and prepares the model for inference. Once all models are set up, the Flask backend server can be launched by navigating to the backend directory and running the command python app.py.
With the server running, the user can access the application through a modern web browser by visiting http://127.0.0.1:5000/. This launches the browser-based interface where all features—text QnA, image captioning, image-based question answering, and speech-to-text—can be accessed.

APPENDIX D: GITHUB REPOSITORY

The full source code, setup instructions, model references, and additional documentation are available on GitHub:
🔗 GitHub Repository:
https://github.com/dhruvbhadhotiya/NOVA
The repository includes:

- Full backend and frontend code
- Installation instructions
- Sample data and screenshots
- Troubleshooting notes

APPENDIX E: TOOLS AND RESOURCES USED

- Development Environment: Visual Studio Code, Jupyter Notebook
- Version Control: Git, GitHub
- API Testing: Postman
- AI Model Hosting & Docs: Hugging Face (for Phi-2, BLIP, speech_recognition)
- Model Optimization: OpenVINO Toolkit (Intel)
- Audio Processing: FFmpeg (optional, for audio enhancement)