

**Project Design Phase-II Technology
Stack (Architecture & Stack)**

| | |
|---------------|---------------------------------|
| Date | 27 OCTOBER 2023 |
| Team ID | Team-592895 |
| Project Name | Lip Reading Using Deep Learning |
| Maximum Marks | 4 Marks |

Technical Architecture:

The Deliverable shall include the architectural diagram as below and the information as per the table1 & table

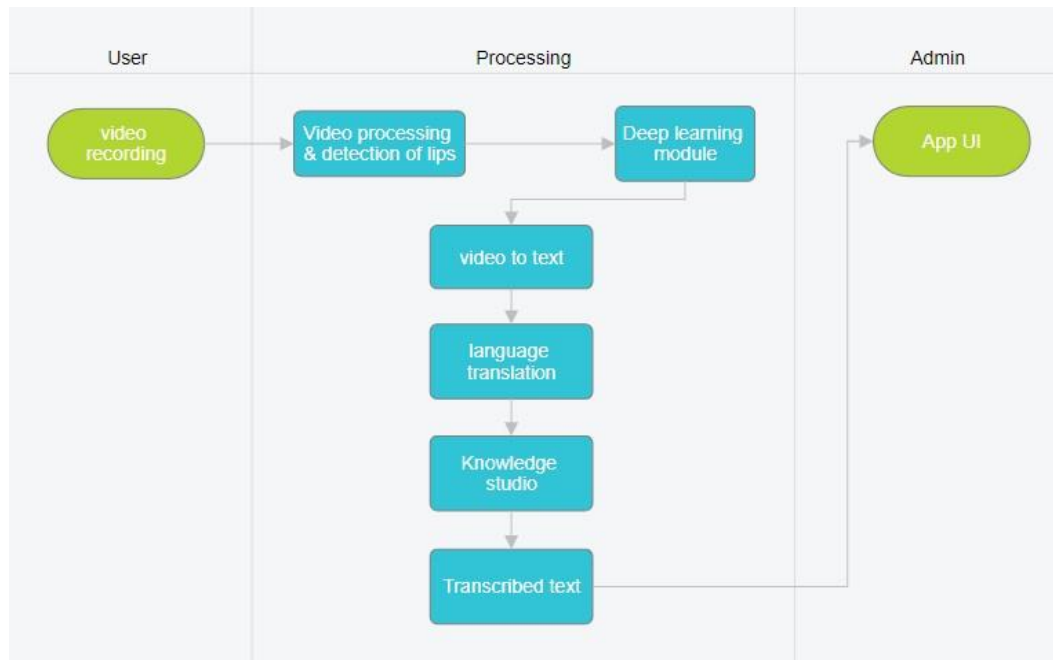


Table-1 : Components & Technologies:

| S.No | Characteristics | Description | Technology |
|------|-------------------------|--|--|
| 1 | Data Collection | Gather a dataset of video clips showing lip movements and corresponding transcriptions. | Video cameras, data labeling tools |
| 2 | Data Preprocessing | Prepare and clean the data for training. This involves video frame extraction, alignment, and cleaning of text data. | Python, OpenCV, Video processing libraries |
| 3 | Feature Extraction | Extract relevant features from lip movements and convert them into a format suitable for deep learning models. | Convolutional Neural Networks (CNN), PyTorch, TensorFlow |
| 4 | Model Development | Create and train deep learning models for lip reading. You may use various architectures like CNNs, RNNs, or Transformer-based models. | PyTorch, TensorFlow, Keras |
| 5 | Training Infrastructure | Utilize hardware resources for training, which can include GPUs or TPUs for faster model training. | NVIDIA GPUs, Google Colab, AWS, Azure |

| | | | |
|---|-----------------------|---|------------------------------|
| 6 | Model Evaluation | Assess model performance using evaluation metrics such as accuracy, Word Error Rate (WER), or Character Error Rate (CER). | Python, Evaluation libraries |
| 7 | Hyperparameter Tuning | Optimize hyperparameters to improve model performance. | Grid search, random search |

| | | | |
|----|-----------------------|---|--|
| 8 | Model Deployment | Deploy the lip-reading model in a production or application environment. | Web frameworks (e.g., Flask, Django), cloud platforms (e.g., AWS, Azure) |
| 9 | Real-time Lip Reading | Develop a real-time lip reading application that captures and processes video streams. | Webcam, OpenCV for real-time video processing |
| 10 | User Interface (UI) | Design a user-friendly interface for the lip reading application. | Front-end frameworks (e.g., React, Angular), HTML/CSS |
| 11 | Database (optional) | Store and manage data, such as user profiles and transcriptions, if required for the application. | Databases (e.g., PostgreSQL, MongoDB) |

| | | | |
|----|----------------------------|---|---|
| 12 | API Integration (optional) | If needed, integrate with external services or APIs for additional functionality. | API libraries, RESTful APIs |
| 13 | Monitoring and Logging | Implement monitoring and logging to track application usage and detect issues. | Logging tools (e.g., ELK stack) |
| 14 | Security | Implement security measures to protect data and user privacy. | Encryption, authentication, authorization |
| 15 | Documentation | Create thorough documentation for the project, including code, usage guides, and model documentation. | Markdown, documentation platforms (e.g., Read the Docs) |

Table-2: Application Characteristics:

| S.No | Characteristics | Description | Technology |
|------|------------------------|--|--|
| 1 | Open-Source Frameworks | Utilizes open-source frameworks for model development and web application. | - Deep learning frameworks: PyTorch, TensorFlow - Web framework (for UI): Flask or Django |

| | | | |
|---|--------------------------|---|--|
| 2 | Security Implementations | Implements security measures to protect data and user privacy, including encryption and access controls. | <ul style="list-style-type: none"> - Encryption (e.g., TLS/SSL for data in transit) - Access controls (e.g., Role-Based Access Control, JWT) - Use of authentication libraries (e.g., OAuth2, JWT tokens) |
| 3 | Scalable Architecture | The application is designed for scalability. The architecture may be a combination of 3-tier or microservices. | <ul style="list-style-type: none"> - Microservices architecture for scalability - Use of containerization (e.g., Docker, Kubernetes) |
| 4 | Availability | Ensures high availability through load balancing and distributed servers. | <ul style="list-style-type: none"> - Use of load balancers (e.g., HAProxy, Nginx) - Distribution across multiple data centers or cloud regions |
| 5 | Performance | Designed for optimal performance, considering the number of requests per second, cache usage, and content delivery networks (CDNs). | <ul style="list-style-type: none"> - Caching mechanisms (e.g., Redis, Memcached) - Content Delivery Network (CDN) for static assets |