BTP PHASE -2

submitted in partial fulfillment of the requirements

for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINNEERING

by

UMAKANT CS21B050

AMAN KUMAR CS21B001

Supervisor(s)

CHALAVADI VISHNU

DEPARTMENT OF COMPUTER SCIENCE AND ENGINNEERING

INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI

MAy 2025

DECLARATION

We declare that this written submission represents our ideas in our own words, and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission to the best of our knowledge. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

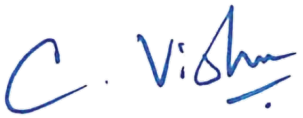
Signature
Umakant
cs21b050

Place: Tirupati
Date: 04-12-2024

Signature
Aman Kumar
cs21b001

BONA FIDE CERTIFICATE

This is to certify that the report titled **VIDEO LLMs**, submitted by **Author(s)**, to the Indian Institute of Technology, Tirupati, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the project work done by them under my supervision. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.



Place: Tirupati
Date: 04-12-2024

Chalavadi Vishnu
Guide
Assistant Professor
Department of Computer Science and Engineering
IIT Tirupati - 517619

ACKNOWLEDGMENTS

We would like to express our heartfelt gratitude to all those who have supported and guided us throughout the completion of the first phase of our B.Tech final year project on "Video LLMs." Firstly, we are deeply grateful to our project guide, Chalavadi Vishnu, for their invaluable guidance, encouragement, and constructive feedback. Their expertise and patience have been instrumental in shaping the direction of this project. We extend our sincere thanks to the faculty and staff of the Department of Computer Science and Engineering (CSE), IIITT, for providing the resources and a conducive environment for research and development. We are also grateful to our peers and team members for their collaboration, insights, and shared learning experiences during this phase. Finally, we would like to thank our families and friends for their unwavering support and motivation, which have been a constant source of strength throughout this journey. This project has been an enriching learning experience, and we look forward to further exploring and contributing to this exciting field in the upcoming phases.

ABSTRACT

In the rapidly growing multimedia content era, understanding and summarizing video data has become an essential task with diverse applications in accessibility, surveillance, and content indexing. Our project aims to build an intelligent system that processes video input from a device's camera to generate a concise textual summary of the video's content.

The proposed system begins by capturing video input and extracting individual frames. A Temporal Convolutional Network (TCN) model filters and identifies significant frames, ensuring computational efficiency and relevance. These significant frames are then processed using a dense captioning model to generate detailed textual descriptions for each frame, capturing key activities and events.

To provide a coherent and meaningful summary of the video, the individual captions are passed through a text summarization module, consolidating the information into a single, concise output describing the overall video content.

This pipeline integrates advanced computer vision and natural language processing techniques to bridge the visual and textual data gap. The system has potential applications in automated video analysis, accessibility tools for visually impaired individuals, and multimedia content management. Future phases of the project will focus on optimizing model performance, allowing metaphor captioning, generating context-specific captions, building a recognition system, and adapting the system to diverse video domains.

TABLE OF CONTENTS

ABBREVIATIONS

The research scholar/ student must take utmost care in the use of technical abbreviations. For example, gms stands for gram meter second not grams. Grams should be abbreviated as g. Abbreviations should be listed in alphabetical orders as shown below.

IITT	Indian Institute of Technology, Tirupati
CSE	Computer Science and Engineering
Dept.	Department of
LLMs	Large Language Models

CHAPTER 1

INTRODUCTION

In the digital age, multimedia content has become a significant aspect of everyday life. With the rapid growth of video data generated across platforms, understanding and summarizing video content in a meaningful way has emerged as a crucial challenge. Manual analysis of videos is both time-consuming and error-prone, underscoring the need for automated systems capable of interpreting and summarizing videos efficiently.

1.1 Motivation

The motivation behind this project is to develop a system that aids not only in summarizing video content but also in making it accessible to visually impaired individuals through audio narration. Automating this process can significantly enhance accessibility,and provide quick insights from video data or live video.

1.2 Problem Statement

Visually impaired individuals struggle to interpret their surroundings in real time due to the lack of comprehensive assistive solutions. Existing technologies provide basic cues but fail to deliver detailed, dynamic scene descriptions.

The Video LLM project addresses this by capturing real-time video, processing key frames, and generating contextual descriptions, which are converted into speech using the Microsoft Edge Text-to-Speech model. By integrating computer vision, NLP, and TTS, this system enhances accessibility and independence for visually impaired users.

CHAPTER 2

Work of Phase-1

we are able to extract the frames and able to generate the description of each frame even the detailed description also

2.1 Frame Extraction

The frame extraction process was implemented using OpenCV. The primary goal of this phase was to extract frames from a video file and save them in a specified folder for further processing, such as video captioning or analysis.

2.1.1 Results of Frame Extraction:

- The process successfully extracted frames from the video.
- Every 10th frame was saved, as specified by the frame interval parameter.
- The frames were stored as PNG images, named in the format `frame_XXXX.png`, where `XXXX` is a 4-digit frame index.
- Example filenames include `frame_0000.png`, `frame_0010.png`, etc.

2.2 Frame Filtration Strategy

This stage focused on refining the extracted frames by removing redundant ones and detecting significant boundary frames. The methodology used ResNet50 as a feature extractor, followed by boundary detection based on cosine similarity.

2.2.1 Results of Frame Filtration:

- Frames with high redundancy were filtered out, reducing storage requirements and processing time.
- Key boundary frames were successfully identified, ensuring that only distinct and meaningful frames were retained for further processing.

```
Extracting frames...
Extracted 185 frames
Extracting features...
Features extracted with shape: (185, 2048)
Detecting boundaries...
Detected 0 boundaries
Saving boundary frames...
Processing complete!
```

Figure 2.1: Result after filtration

2.3 Image Captioning Using BLIP-2

The BLIP-2 (Bootstrapping Language-Image Pretraining) model was usedgenerate captions for selected frames. This model is widely recognized for its effectiveness in generating accurate and meaningful images descriptions.

2.3.1 Results of Image Captioning:

- BLIP-2 successfully generated captions for the extracted frames, improving video understanding.
- Captions were overlaid onto images and saved for further analysis.

Figure 2.2: Result after captioning

2.4 Detailed Image Captioning Using Florence-2

In addition to BLIP-2, we experimented with the Florence-2 model from Microsoft, designed for advanced vision-language tasks. This model was used to generate detailed captions based on given text prompts.

2.4.1 Results of Detailed Captioning:

- Florence-2 generated highly detailed and contextually rich captions for selected frames.
- The captions provided a deeper understanding of objects, actions, and interactions within each frame.

The image is a black and white photograph of three people in a small boat on the water. The boat appears to be made of wood and is small in size. The people in the boat are dressed in traditional Indian clothing, with the man on the left wearing a turban, the man in the middle wearing a long robe, and the woman on the right wearing a headscarf. They are all holding black umbrellas and appear to be rowing the boat.the center of the boat, there is a large wooden bench with a cushion on it. The bench is empty and there are a few items scattered around it. On the right side of the bench, there are two people sitting on the bench and one person is holding a paddle. The background is a cloudy sky and the water is calm. The overall mood of the image is peaceful and serene.

CHAPTER 3

Work of Phase-2: Text Summarization

3.1 Model: BART (facebook/bart-large-cnn)

For text summarization, we employed BART, a transformer-based model designed for sequence-to-sequence tasks like summarization, text generation, and translation. It works as a denoising autoencoder, reconstructing the original document from a corrupted input.

3.2 How the Model Works

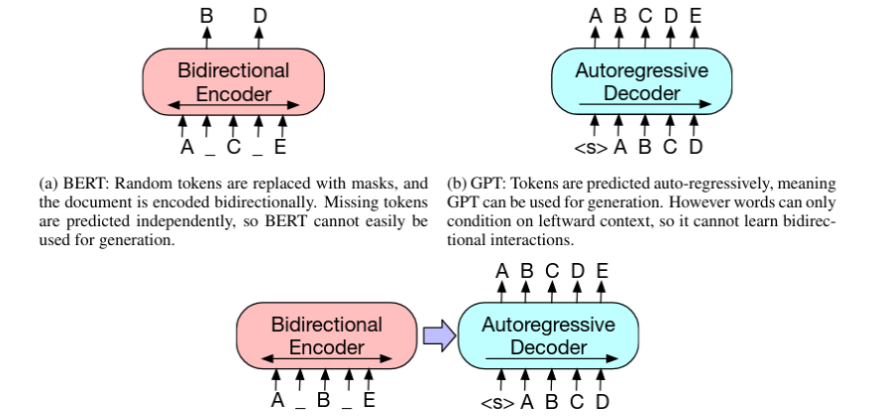
- Preprocessing: The input video is first processed to extract textual captions using BLIP-2 and Florence-2 models (from Phase-1).
- Encoding: The BART model takes the extracted text and encodes it using a bidirectional encoder, which understands the context of the input.
- Decoding: The decoder generates a summarized version of the text auto regressively, meaning each generated word depends on previously generated words.

3.3 Architecture of BART

- Encoder-Decoder Structure: Similar to traditional Transformer models, BART consists of an encoder (which processes the corrupted text) and a decoder (which reconstructs the original or summarized text).
- Bidirectional Encoder: Unlike GPT, BART's encoder reads the entire input sequence in both directions to understand context effectively.
- Autoregressive Decoder: The decoder generates output step by step, predicting the next word based on previously generated words.
- Cross-Attention: Each layer in the decoder attends to the encoder's final hidden states, ensuring that the generated summary remains contextually relevant.
- Activation Functions: Uses GELU (Gaussian Error Linear Unit) instead of ReLU for smoother training.

3.4 Results of Text Summarization

- The model successfully generated concise and meaningful summaries of the video's content.



(a) BERT: Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, so BERT cannot easily be used for generation.

(b) GPT: Tokens are predicted auto-regressively, meaning GPT can be used for generation. However words can only condition on leftward context, so it cannot learn bidirectional interactions.

(c) BART: Inputs to the encoder need not be aligned with decoder outputs, allowing arbitary noise transformations. Here, a document has been corrupted by replacing spans of text with mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder. For fine-tuning, an uncorrupted document is input to both the encoder and decoder, and we use representations from the final hidden state of the decoder.

Figure 3.1: Architecture of BART

- Key events from the video were effectively retained while removing redundant information.
- The summaries were further converted into audio format, making them accessible to visually impaired individuals.

Video Description:
There are three people in a boat with umbrellas on the water. Moving to another group. Then, A couple in traditional Indian attire posing for a picture. Continuing the narrative, Subsequently, There are two women standing on a balcony looking at something. Continuing the narrative, Following that, Several people are standing around a woman in a sari. Continuing the narrative, In another scene, Bride and groom in traditional Indian attire standing together in a crowd. Continuing the narrative, Meanwhile, There is a man and a woman walking up a flight of stairs.

Figure 3.2: Result after Text summarization

3.5 Text to Speech

- Implemented text-to-audio conversion using the Google Text-to-Speech model to assist visually impaired individuals by providing audio descriptions of their surroundings.
- The TTS system will convert generated textual summaries into natural-sounding speech, enhancing accessibility and ensuring that visually impaired users can understand video content effortlessly.

3.6 Architecture

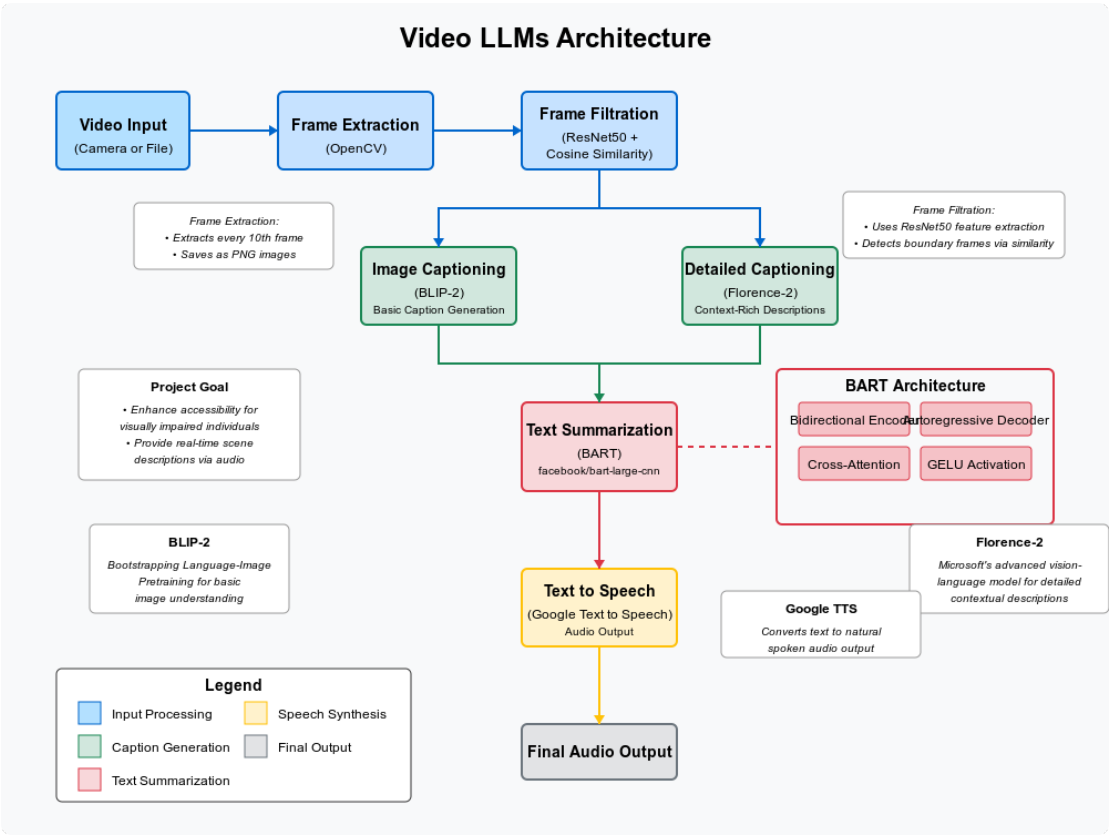


Figure 3.3: Architecture

Conclusion

In this project, we developed a comprehensive AI-driven assistive system aimed at enhancing accessibility for visually impaired individuals through real-time video understanding. The system integrates multiple advanced components—frame extraction, frame filtration, image captioning, text summarization, and text-to-speech conversion—each designed to work cohesively for end-to-end scene interpretation.

The initial phase involved extracting frames from video input using OpenCV, with a strategy to capture every 10th frame to reduce computational load while preserving temporal context. Subsequently, redundant frames were removed using a ResNet50-based feature extraction pipeline and cosine similarity, ensuring that only distinct and meaningful transitions were retained for further analysis.

For semantic understanding, we utilized two powerful vision-language models: BLIP-2 and Florence-2. BLIP-2 generated concise and relevant captions for the filtered frames, while Florence-2 offered more detailed and context-rich interpretations. This dual-model strategy enhanced the overall descriptiveness of the visual data, making it more informative for subsequent processing stages.

To further refine the output, we employed the BART (facebook/bart-large-cnn) model for abstractive summarization. BART effectively distilled the captioned content into coherent summaries that retained key events and removed redundancy. The use of a transformer-based encoder-decoder architecture, with bidirectional encoding and autoregressive decoding, allowed for contextually accurate and linguistically fluent summaries.

Finally, the processed textual summaries were converted into speech using the Microsoft Edge Text-to-Speech engine. This step completed the video-to-audio pipeline, enabling real-time auditory feedback for the user. The use of natural-sounding speech ensured a seamless and intuitive experience, significantly improving the user's ability to perceive and understand their environment.

Overall, this project demonstrates a robust integration of computer vision, natural language processing, and speech synthesis technologies. It presents a scalable and effective solution for real-time video interpretation, with the potential to greatly enhance the independence and quality of life for visually impaired users. Future work may include optimizing the pipeline for edge deployment, expanding language support, and incorporating user feedback mechanisms to further tailor the system to individual needs.

3.7 Refrences

- https://arxiv.org/pdf/2311.15879
- https://arxiv.org/pdf/1910.13461

CHAPTER 4

Internship Work: Code Configurator

The Code Configurator is a low-code framework designed to empower non-developers to seamlessly configure and integrate third-party insurance products with the company's systems, eliminating the need for extensive technical expertise.

4.1 Problem statement

Integrating third-party insurance products into the company's systems is currently a time-consuming process, typically requiring 10–12 days of effort from software developers. This manual approach not only delays time-to-market but also demands significant technical involvement for each integration.

To enhance productivity and reduce dependency on developers, there is a need for a solution that allows non-developers to configure and integrate insurance products independently within 1–2 days. This would significantly streamline the integration process, improve operational efficiency, and accelerate product deployment.

4.2 Research on Previous Integrations

As part of our research, we conducted an in-depth analysis of existing third-party insurance product integrations. The primary objective was to assess the feasibility of configuring these integrations through a low-code or no-code approach.

Our findings revealed a consistent and modular pattern across most integration flows. Specifically:

- Each product can be broken down into multiple **stages**.
- Each stage comprises several **flows**, representing the logical sequence of operations.

To accommodate this structure, the proposed framework is designed to be flexible and extensible, allowing dynamic chaining of stages and flows, regardless of their complexity or number.

Additionally, we identified the need for a standardized, language way to define the integration logic. As a result, we introduced a custom JSON Expression Language, which allows the logic for each flow to be expressed in a structured JSON format. This expression language can be easily parsed and evaluated by a dedicated evaluator, enabling seamless configuration and execution without requiring traditional coding skills.

- Defined a structured approach to represent API attributes using JSON Expression Language (JEL).
- Mapped API parameters and configurations into JEL to ensure dynamic and flexible integration.

4.3 Json Expression Language

We define the integration logic using a tree-like structure composed of individual units called nodes. Each node represents a specific operation or condition within the integration flow.

The evaluation process begins at the root node and proceeds recursively down the tree, traversing through each child node. This continues until the evaluator reaches and processes the leaf nodes, effectively completing the logic defined for that flow.

This hierarchical structure allows for clear, modular, and scalable representation of complex logic, making it easier to visualize and manage integration workflows.

4.4 Node Types and Their Structures

To define the integration logic in a flexible and modular way, we utilize a tree-like structure composed of different node types. Each node serves a specific purpose in the evaluation flow. Below are the primary node types, their purposes, structures, and examples.

4.4.1 1. Data Node

Purpose: Stores static or directly mapped values.

Structure:

```
{
  "type": "Data_Node",
  "sub_type": "hardcoded" | "direct",
  "value": "<value>"
}
```

Sub-Types:

- **Hardcoded** – Represents static values.
- **Direct** – Directly maps from the input payload.

Examples:

```
// Hardcoded example
{
  "type": "Data_Node",
  "sub_type": "Hardcoded",
  "value": "Father In Law"
}

// Direct map example
{
  "type": "Data_Node",
  "sub_type": "Direct",
  "value": "data.<field>"
}
```

4.4.2 2. Operator Node

Purpose: Performs operations such as formatting, logical comparisons, conditionals, arithmetic, hashing, and array lookups.

Structure:

```
{
  "type": "operator_node",
  "sub_type": "concat" | "toString" | "formatDate",
  "arguments": ["<operation>"],
  "children": [
    {
      "type": "Data_Node",
      "sub_type": "Hardcoded" | "Direct" | "Config",
      "value": "<value>"
    },
    {
      "type": "Data_Node",
      "sub_type": "Hardcoded" | "Direct" | "Config",
      "value": "key1.key2"
    }
  ]
}
```

Example: Concatenation of Tax Values

```
{
  "type": "operator_node",
  "sub_type": "concat",
  "operator": "_",
  "children": [
    {
      "type": "Data_Node",
      "sub_type": "Direct",
      "value": "response.CGSTTaxPremium"
    },
    {
      "type": "Data_Node",
```

```
      "sub_type": "Direct",
      "value": "response.SGSTTaxPremium"
    }
  ]
}
```

4.4.3 3. Switch Node

Purpose: Implements conditional logic to choose between multiple branches based on evaluation results.

Structure:

```
{
  "type": "switch_node",
  "calculation": "<condition_expression>",
  "branch": {
    "x1": { <result_if_x1> },
    "x2": { <result_if_x2> }
  }
}
```

Example: Handling Loan Tenure Units

```
{
  "type": "switch_node",
  "calculation":{
    "type": "Operator",
    "sub_type": "toLowerCase",
    "children": [
    {
    "type": "data",
    "sub_type": "direct",
    "value": "payload.data.loan_details.loan_tenure_unit"
    }
```

```
      ]
    }
    "branch": {
      "true": {
        "type": "operator_node",
        "sub_type": "arithmetic",
        "operator": "divide",
        "children": [
          {
            "type": "Data_Node",
            "sub_type": "Direct",
            "value": "loanDetails?.loan_tenure"
          },
          {
            "type": "Data_Node",
            "sub_type": "Hardcoded",
            "value": "12"
          }
        ]
      },
      "false": {
        "type": "Data_Node",
        "sub_type": "Direct",
        "value": "loanDetails?.loan_tenure"
      }
    }
}
```

4.5 Stages and Flows

4.5.1 Stage Expression

The **Stage** in an expression defines the context or lifecycle step of the insurance product. The supported stages are:

- **X**
- **Y**
- **Z**
- **W**

Each stage comprises one or more **flows**, where each flow represents an API call or action to be performed within that stage.

4.5.2 Flow Definition

Each flow in a stage includes the following components:

1. **name**
   The name of the API.
2. **url**
   API endpoint expressed as a `Data Node` with a direct mapping from configuration.
3. **method**
   HTTP method (e.g., GET, POST) expressed as a `Data Node` with a hardcoded value.
4. **entry_condition**
   A node expression that returns a boolean value to determine whether this API should be called or skipped.
5. **headers**
   API headers represented using node structures.
6. **input_mapping**
   The API request payload, structured using nodes.
7. **requestValidations** and **responseValidations**
   These validations determine whether the input or output contains errors. Expressed as an array of `switch_node` elements. Each switch node evaluates a condition; if true, the corresponding error message is returned; if false, an empty string is returned.
   Structure:

```
{
  "requestValidations" or "responseValidations": [{
    "type": "switch_node",
    "calculation": "<condition_expression>",
    "branch": {
```

```
        "true": { <error message node> },
        "false": {
          "type": "Data_Node",
          "sub_type": "Hardcoded",
          "value": ""
        }
      }
    }]
  }
```

8. **errorMapping**
   Describes how error responses should be structured. It uses standardized keys and maps them to validations evaluated as true.
   Structure:

```
{
  "status": {
    "type": "Data_Node",
    "sub_type": "Hardcoded",
    "value": "failure"
  },
  "validationMessages": {
    "type": "Data_Node",
    "sub_type": "Direct",
    "value": "payload.requestValidations | payload.responseVal
  }
}
```

9. **successMapping**
   Specifies how to extract only the necessary data from a successful flow. This is used to shape the final gateway output. Only pre-approved (whitelisted) keys are permitted in this mapping. Each stage defines its own set of standardized output keys.

4.6 Evaluator Engine Architecture

Figure 5.1 illustrates the architecture of the Evaluator Engine, which is responsible for executing flows based on the provided stage, product, and partner configurations.

Execution Flow Description

1. **Service_X Input Invocation**
   The execution starts with a request made to `/execute` endpoint in **Service_X**. The parameters passed include stage,flow,product

2. **Delegation to Service_Y**
   Upon receiving the request, **Service_X** delegates the processing responsibility to **Service_Y**.
3. **Configuration Lookup**
   In **Service_Y**, for each flow under the given stage, configuration is looked up.It first attempts to retrieve the Input/Output mapping from **Redis**.If a cache **hit** occurs, it proceeds with the cached configuration.If a cache **miss** occurs, it retrieves the configuration from the **Configurator Database**.
4. **Flow Execution**
   The selected or loaded configuration is then used to execute the flow logic as defined under the current stage.
5. **Response Construction**
   After successful execution of all flows, the **Standard Gateway Output** is constructed and sent back to **Service_X** as the response.

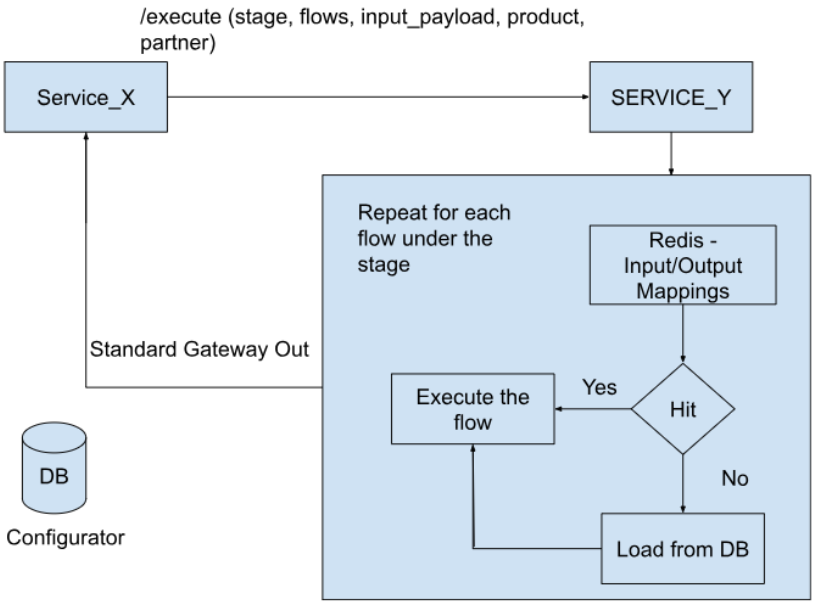


Figure 4.1: Evaluator Engine

Key Components

- **Service_X**: Initiates the evaluation request and receives the final gateway output.
- **Service_Y**: Responsible for evaluating the flows for the given stage using configuration.
- **Redis**: Acts as a cache for flow configuration (input/output mappings).
- **Configurator Database (DB)**: Persistent storage for configuration in case Redis misses.
- **Flows**: A sequence of defined API executions evaluated within a stage context.

CHAPTER 5

Internship Work: Mentormatch

5.1 Problem Statement

In today's rapidly evolving professional landscape, finding the right mentor or mentee can be challenging. Traditional networking methods often lack structure, accessibility, and efficient matching mechanisms. The Mentorship Platform addresses these challenges by providing a digital space where professionals can connect, share knowledge, and grow together through structured mentorship relationships.

5.2 Project Overview

The Mentorship Platform is a full-stack web application designed to facilitate meaningful mentor-mentee connections. The platform enables users to create detailed profiles, discover potential mentors or mentees, establish connections, and engage in structured mentorship relationships.

5.3 Technical Architecture

5.3.1 Frontend Implementation

The frontend is built using vanilla JavaScript, HTML5, and CSS3, emphasizing simplicity and maintainability. Key features include:

- Responsive design for cross-device compatibility
- Client-side routing for seamless navigation
- JWT-based authentication system
- Real-time profile updates and connection management

5.3.2 Backend Architecture

The backend is implemented using Node.js and Express.js, providing a robust RESTful API. The system architecture includes:

- RESTful API endpoints for user management
- MySQL database with Sequelize ORM
- JWT-based authentication middleware
- Role-based access control

5.4 Key Challenges and Solutions

5.4.1 User Connection Management

Challenge

Implementing a bidirectional connection system where users can be both mentors and mentees while maintaining data integrity and relationship clarity.

Solution

- Developed a flexible Connection model with mentorId and menteeId
- Implemented status tracking (pending, accepted, rejected)
- Created role validation middleware
- Added connection request management system

5.4.2 Profile Data Structure

Challenge

Designing a profile system that accommodates both mentors and mentees with different requirements while maintaining data consistency.

Solution

- Created a unified Profile model with role-specific fields
- Implemented flexible skills and interests arrays
- Used JSON fields for extensibility

- Added validation for required fields based on user role

5.4.3 Security Implementation

Challenge

Ensuring secure communication and data protection while maintaining user privacy.

Solution

- Implemented JWT-based authentication
- Added CORS configuration with whitelisted origins
- Created middleware for route protection
- Implemented input validation and sanitization

5.5 Advanced Features

5.5.1 Real-time Chat System

The platform includes an integrated chat system that enables:

- Real-time messaging between mentors and mentees
- Message persistence and history
- Read receipts and typing indicators
- File sharing capabilities

5.6 Future Enhancements

5.6.1 Planned Features

- Smart matching algorithm by analyzing preference
- Video conferencing integration for virtual meetings
- AI-powered mentor-mentee matching
- Resource sharing and document management
- Mobile application development

5.7 Technical Achievements

5.7.1 Performance Optimization

- Implemented efficient database queries
- Added client-side caching
- Optimized API response times
- Reduced server load through smart request handling

5.7.2 Scalability Considerations

- Modular architecture for easy scaling
- Database optimization for large datasets
- Load balancing preparation
- Microservices architecture planning

5.8 Conclusion

The Mentorship Platform demonstrates the successful implementation of a modern web application that addresses real-world needs in professional development. Through careful architecture design and implementation of advanced features, the platform provides a robust solution for connecting mentors and mentees while ensuring security, scalability, and user experience.

5.9 Lessons Learned

- Importance of proper authentication and authorization
- Value of modular and maintainable code structure
- Significance of user experience in platform adoption
- Need for scalable database design
- Importance of real-time features in modern web applications