CSC490 Assignment 2

Team Members
Minh Le (1007894432), Prashanth Shyamala (1008819021),
Shaan Purewal (1008332939), York Ng (1009153577)

1 Part One: Aspirational Datasets

To describe our ideal datasets, we first need to describe their use cases for our project. To recap, LectureClip is a system that retrieves and assemble lecture videos so that students can locate the most relevant moments and stitch them into a coherent "sub-lecture." We would then require datasets to test our infrastructure as well as to evaluate our machine learning models that are used for various tasks including transcription, video chunking, topic modeling, and more. The datasets below are therefore organized around these two needs: running the system end-to-end and evaluating its successes in improving students' learninge experience.

Aspirational Datasets. To make LectureClip successful, the *ideal datasets* are defined by what they must enable: (1) a baseline experience (keyword search over a timestamped video + transcript), (2) LectureClip's main features (clip queries, selection, and assembly), and (3) evaluation of both.

Concretely, the minimum viable dataset for **each lecture** is: a raw lecture video, basic lecture metadata, and a **timestamp-aligned transcript segmented into short units** (start/end times + text). This baseline is necessary because both our baseline system (keyword transcript search with timestamped matches) and LectureClip's retrieval operate over the same time-indexed text backbone. In an *ideal* setting, the lecture dataset would additionally include aligned cues that improve clip boundary selection and retrieval robustness, such as timestamps for slide change, OCR text extracted from slides/board, and markers of speaker/topic transitions. These cues are not required to run the system, but they make our task easier by enabling cleaner boundaries and improving the coherence of assembled sub-lectures.

Aspirational Evaluation Benchmarks. Although LectureClip will primarily use off-the-shelf models for retrieval and summarization, we still require a measurable definition of project success. Our core claim is that LectureClip improves the lecture-watching experience by helping students locate relevant content more quickly than existing tools and assemble that content into a coherent sub-lecture that addresses their query. Given this, we evaluate our models against a baseline that approximates common lecture software, namely keyword-based transcript search with timestamped matches and manual scrubbing, using three benchmarks.

1. **Task efficiency benchmark**: measures time-to-first-relevant content, time-to-sufficient understanding, and interaction effort.
2. **Offline retrieval quality benchmark**: uses fixed queries and human-labeled relevant time spans to compute ranking metrics such as Precision@k and nDCG@k based on overlap with predicted contiguous clips.
3. **Sub-lecture quality benchmark**: assesses the coherence and usefulness of the assembled sequence using a human rubric (relevance, coverage, coherence or flow, redundancy) and, optionally, blind pairwise preference tests against the baseline.

These benchmarks will let us measure whether LectureClip delivers practical improvements over the baseline in two areas: *speed* (students reach relevant content with less time and effort) and *quality* (the retrieved spans and assembled sub-lecture are accurate and sufficiently cover the query).

2 Part Two: Available Datasets

Relevant item	Source (link)	Description of the data	Commentary
MIT OCW / Open Learning Library lectures	Open Learning Library	Lecture videos + course metadata; many pages provide downloadable transcripts/captions (SRT/TXT).	This dataset contains video + metadata + timestamped transcript; will be used for baseline keyword search and LectureClip retrieval. Scraping transcripts may be needed.
MaViLS (public research dataset)	MaViLS (GitHub)	Lecture videos with slides + transcripts and alignment signals linking slides, video, and spoken content.	This dataset provides a benchmark for slide change / slide alignment; will be used to improve clip boundary selection and reduce irrelevant seconds in contiguous clips.
M3AV (public research dataset)	M3AV (GitHub)	Academic lecture videos with high-quality speech transcripts and OCR labels (including complex text).	This dataset can be used to evaluate OCR services to make retrieval more robust and help segment and assemble cleaner sub-lectures. This dataset may also be used to benchmark clip retrieval quality
QVHighlights (public benchmark for query → moments)	QVHighlights (Moment-DETR repo)	Natural language queries paired with ground-truth relevant moments and graded saliency scores.	This dataset supports offline retrieval evaluation by providing immediately usable queries + labeled time spans for overlap-based metrics and ranking metrics such as Precision@k and nDCG.
ActivityNet Captions (public benchmark for text → temporal segments)	ActivityNet Captions (HF)	Videos paired with temporally localized sentence descriptions (each sentence maps to a time segment).	This dataset is not lecture-specific, but it is useful for prototyping and evaluating query → temporal segment retrieval with overlap-based metrics. We can use it as a sanity check before testing our model on lecture videos.
Team-curated LectureClip eval set (manual)	Drive link to manually scrapped UofT Lectures	A small lecture subset where the team will create: fixed queries, ground truth relevant topic spans, sub-lecture quality ratings, and task efficiency logs.	The team has collected a curated subset of lecture recordings, ensuring each lecture includes basic metadata and a timestamped transcript (if applicable). We also will create manual annotations over transcript segments, including subject/topic boundaries and labeled relevant time spans to support retrieval and clip assembly evaluation. The segment labels will be manually produced, and the task efficiency metrics require a small user study.

Table 1: Relevant datasets for LectureClip

3 Part Three: Data-processing Pipelines

LectureClip consists of two data processing pipelines: an **ingestion pipeline** triggered on video upload, and a **query pipeline** triggered on student search. Both are exposed via Amazon API Gateway and orchestrated by AWS Lambda.

3.1 System Architecture

Figure 1 shows the end-to-end architecture. The `/upload` path handles transcription, chunking, and embedding, writing results to four storage backends; the user receives a unique `lecture_id` as a key. Given the user-supplied `lecture_id` and topic query, the `/query` path then embeds the user query, performs semantic retrieval, and assembles the final video.

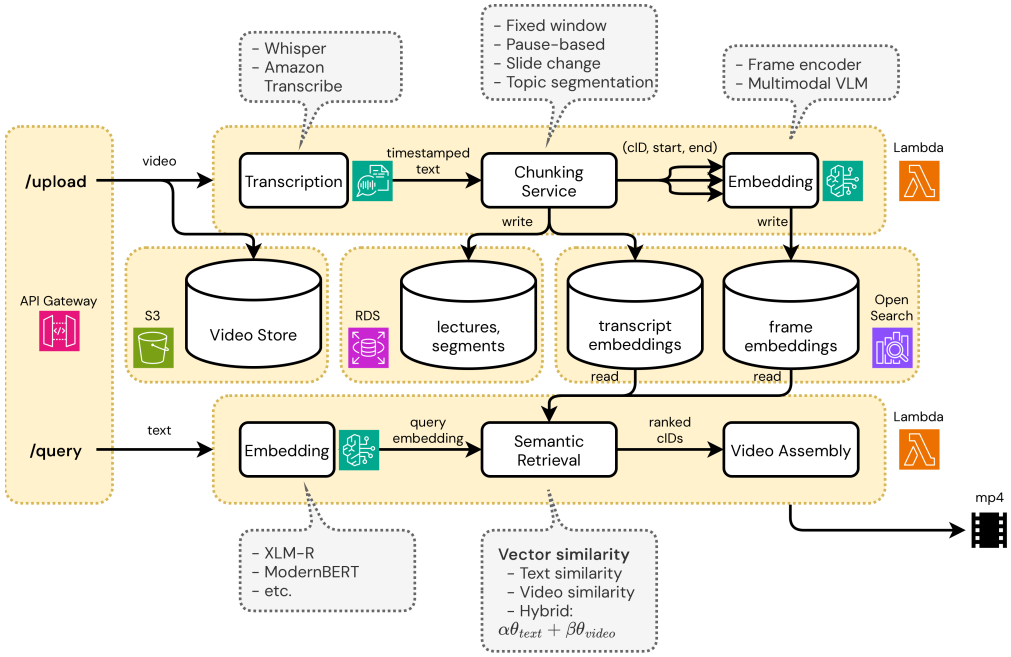

Figure 1: LectureClip System Architecture

Ingestion pipeline (`/upload`)

Transcription. The video is stored in S3 and submitted to Amazon Transcribe (or other solutions such as Whisper). Transcribe returns a JSON containing each sentence and its `(start_s, end_s)` interval.

Chunking. A second Lambda splits the transcript into chunks and assigns each a `segment_id`. We currently implement **sentence chunking**: multiple sentences are grouped together once they reach a certain character threshold, ensuring each segment has a similar length. The `(segment_id, lecture_id, start_s, end_s, transcript)` tuples are written to the `segments` and Transcripts tables in RDS. In the future, we aim to experiment with other chunking implementations, which are discussed in Section 3.3.

Embedding. Each chunk's transcript text is embedded using Amazon Bedrock Titan Embeddings, and the visual content (from key frames sampled from the source video) are embedded using a multimodal model via Bedrock, which we persist in OpenSearch.

Query pipeline (`/query`)

Query embedding. The user's free-text topic is embedded with the same Bedrock Titan Embeddings v2 model used at ingestion time.

Semantic retrieval. The query vector is matched against the OpenSearch index. Beyond simple cosine similarity, we plan to train a dedicated segment-ranking model (see Section 3.3) that scores each candidate chunk based on the text and visual embeddings' relationship to the topic embedding.

Video assembly. The top-k ranked `cID`s are resolved to `(vID, start_s, end_s)` spans via RDS. Spans are sorted by start time, overlapping windows are merged, and segments are concatenated. The assembled MP4 is written back to S3 and returned as a pre-signed URL.

3.2 Data Schemas

Raw lecture videos and large derived artifacts (e.g., OCR outputs, slide-change timelines, detailed interaction logs) will be stored in **S3**. Structured metadata, transcript segments, queries, and human labels should be stored in **PostgreSQL** using **RDS** for reliable joins and reproducible evaluation, while vector embeddings will be stored in **OpenSearch**.

1. **Lecture metadata (PostgreSQL; video in S3)**

```
CREATE TABLE lectures (
  lecture_id    UUID PRIMARY KEY,
  course_id     TEXT,
  title         TEXT,
  duration_s    REAL,
  video_uri     TEXT NOT NULL,      -- S3 URI
  ingested_ts   TIMESTAMPTZ NOT NULL DEFAULT now()
);
```

2. **Timestamped transcript segments (PostgreSQL)**

```
CREATE TABLE segments (
  segment_id  UUID PRIMARY KEY,
  lecture_id  UUID NOT NULL REFERENCES lectures(lecture_id),
  idx         INT  NOT NULL,
  start_s     REAL NOT NULL,
  end_s       REAL NOT NULL,
  text        TEXT NOT NULL,
  UNIQUE (lecture_id, idx)
);
```

3. **Fixed evaluation queries (PostgreSQL)**

```
CREATE TABLE eval_queries (
  query_id     UUID PRIMARY KEY,
  lecture_id   UUID NOT NULL REFERENCES lectures(lecture_id),
  query_text   TEXT NOT NULL
);
```

4. **Ground-truth relevant spans per query (PostgreSQL)**

```
CREATE TABLE truth_spans (
  span_id     UUID PRIMARY KEY,
  query_id    UUID NOT NULL REFERENCES eval_queries(query_id),
  start_s     REAL NOT NULL,
  end_s       REAL NOT NULL,
  relevance   SMALLINT NOT NULL  -- 2=relevant, 1=partial, 0=non
);
```

5. **Task efficiency outcomes (PostgreSQL; logs optionally in S3)**

```
CREATE TABLE eval_task_runs (
  run_id        UUID PRIMARY KEY,
  query_id      UUID NOT NULL REFERENCES eval_queries(query_id),
```

```
  condition     TEXT NOT NULL,      -- 'baseline' or 'lectureclip'
  participant_id TEXT,              -- optional (hashed/pseudonymous)
  time_to_first_relevant_s REAL,
  time_to_understanding_s  REAL,
  interaction_count INT,
  interaction_log_uri TEXT           -- optional S3 URI to detailed logs
);
```

6. **Sub-lecture quality judgments (PostgreSQL)**

```
CREATE TABLE sublecture_ratings (
  rating_id   UUID PRIMARY KEY,
  query_id    UUID NOT NULL REFERENCES eval_queries(query_id),
  condition   TEXT NOT NULL,        -- 'baseline' or 'lectureclip'
  rater_id    TEXT,                 -- optional (hashed/pseudonymous)
  relevance   SMALLINT NOT NULL,     -- e.g., 1-5
  coverage    SMALLINT NOT NULL,
  coherence   SMALLINT NOT NULL,
  redundancy  SMALLINT NOT NULL,
  preferred   BOOLEAN,              -- optional pairwise preference
  notes       TEXT
);
```

3.3 Next Steps

Storing embeddings in OpenSearch. The immediate blocker is an IAM permissions issue preventing Lambda from writing to the OpenSearch domain. Once resolved, each chunk's text vector (and later its frame vectors) will be indexed under its `segment_id`.

Alternative chunking strategies. Fixed interval chunking serves as a baseline, but doesn't capture topic boundaries. We plan to explore (a) *pause-based* splitting on silence gaps from the Transcribe output, (b) *slide-change* splitting via frame differencing, and possibly (c) *topic-segmentation* chunking using cosine-drift over sentence embeddings (e.g., TextTiling). Each strategy will be a separate Lambda handler selectable at upload time.

Visual (frame) embeddings. Sampled video frames will be embedded using a multimodal model on Bedrock (e.g. Nova), quantifying the visual information in each segment alongside the transcript. This is particularly useful for lectures where the most informative content are in diagrams or graphics.

Learned segment-ranking model. Rather than ranking retrieved chunks by cosine similarity alone, we plan to train a lightweight ranking model that scores each candidate segment given the query. For a query q and candidate chunk c, the model takes three inputs: (1) the topic embedding $\mathbf{e}_q$, (2) the chunk's transcript embedding $\mathbf{e}_{\text{text}}$, and (3) the chunk's frame embedding $\mathbf{e}_{\text{frame}}$. These are concatenated and passed through an MLP/regression model to produce a scalar relevance score $s(q, c) \in \mathbb{R}$, trained with a pairwise ranking loss objective.

This training will require a manually curated chunk ranking dataset: for each evaluation query, we will rank candidate chunks. We will produce this from our UofT lecture set, storing labels in the `chunk_rankings` table above. A held-out test split serves double duty: it directly evaluates our ranking model via nDCG@k and Precision@k, and it also lets us benchmark alternative transcript and frame embedding models by swapping them in, running the same query, and evaluating the quality of the segment rankings.

4 Part Four: Infrastructure as Code Implementation

Our system implementation is stored in two separate GitHub repositories, one containing the app and the other containing our infrastructure. They are LectureClip-App and LectureClip-Infra, respectively.

5 Part Five: Disaster Recovery Demonstration

We have the video demonstrating our recovery process here: DEMO VIDEO.