Data Preparation/Feature Engineering

Project Title: Lecture Companion: AI-Powered Translation and Summarization tool for Burmese Learners

Team Members

1. Lwin Naing Kyaw

2. Nang Hom Paung

3. Sai Aike Sam

4. Hom Nan Thawe Htun

5. MangShang SauYing

6. Ingyin Khin

1. Overview

The data preparation phase for the "Lecture Companion" project is a multi-stage pipeline designed to transform unstructured, real-world educational content into a structured, multi-modal, and analysis-ready format. The primary goal is to ingest raw lecture videos (or their transcripts), process them into clean text, and then engineer specific data representations, such as vector embeddings for retrieval and character-level features for classification, to power the downstream machine learning models. This phase is critical for bridging the gap between raw, noisy data (like lecture audio) and the high-quality, structured inputs required for translation, summarization, and interactive Q&A.

2. Data Collection

The dataset for this project was sourced from open, public-access educational platforms, which are highly representative of the content our target users (Burmese students) would use.

1. **Primary Source: YouTube**. Specifically, lectures from Andrew Ng's Stanford CS229 Machine Learning course (e.g., Lecture 6 - Support Vector Machines) were used as the primary testbed for this report.
2. **Supplementary Sources:** Transcripts from other **MOOC platforms (Coursera, edX)** and **podcasts** were identified to ensure the pipeline is robust to different content styles, as noted in the "Data Research" report.
3. **Collection Method:** A robust data ingestion script was developed. The primary method uses yt-dlp to download the audio (.mp3) directly from a YouTube URL. A critical fallback mechanism is included: if audio download fails (e.g., due to bot detection), the system automatically attempts to fetch the public-facing English captions using the youtube-transcript-api.

3. Data Cleaning

Data cleaning is integrated directly into the pipelines.

1. **Audio-to-Text (ASR):** The faster-whisper model itself acts as a primary cleaning step, normalizing varied speaker accents, handling background noise, and converting spoken audio into a preliminary text transcript.
2. **Text Normalization:** For the RAG pipeline (Pipeline 2), a normalize_spaces function is used to clean the transcript by collapsing all whitespace (tabs, newlines) into single spaces.
3. **Vocabulary Extraction:** For the CEFR classifier (Pipeline 3), the text is converted to lowercase, and a regex (re.findall(r'\b[a-z]{2,}\b', raw_text)) is applied to extract only alphabetic words of two or more letters, effectively filtering out punctuation, numbers, and single-letter artifacts.

4. Exploratory Data Analysis (EDA)

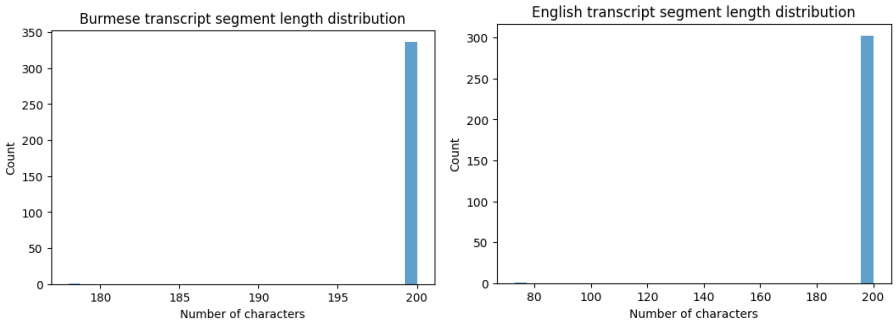
EDA was performed on the outputs of the data preparation pipelines to validate the data's characteristics and suitability for the models.

1. **Key Insights:**

- **Caption Unavailability:** A key finding was that many high-quality YouTube lectures lack official, accurate captions, validating the decision to build the pipeline around a robust ASR model (Whisper) as a necessary fallback.
- **Segment Suitability:** The text chunking strategy (500 characters, 120 overlap) was validated by the "Distribution of Segment Durations" analysis (Figure 4, Data Research Report), which showed most text segments are 8-18 seconds long—an ideal size for low-latency API calls to Gemini.
- **Topical Cohesion:** The "Segment Neighborhood Cohesion" plot (Figure 4, Data Research Report) showed high cosine similarity (0.7-0.8) between adjacent segments, confirming strong topical continuity. This is crucial for the RAG system, as it ensures retrieved context is coherent.

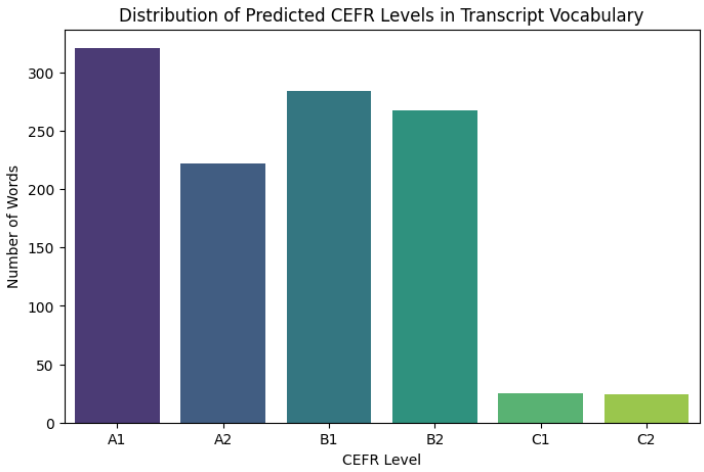
2. **Visualizations:**

- **Text Length Distributions:** Histograms were generated (Pipeline 1 code) to analyze the distribution of text lengths for English and Burmese transcripts. This confirmed the expectation that Burmese text is often longer in characters than the English source, as seen in the "Length Comparison" output (Pipeline 1).



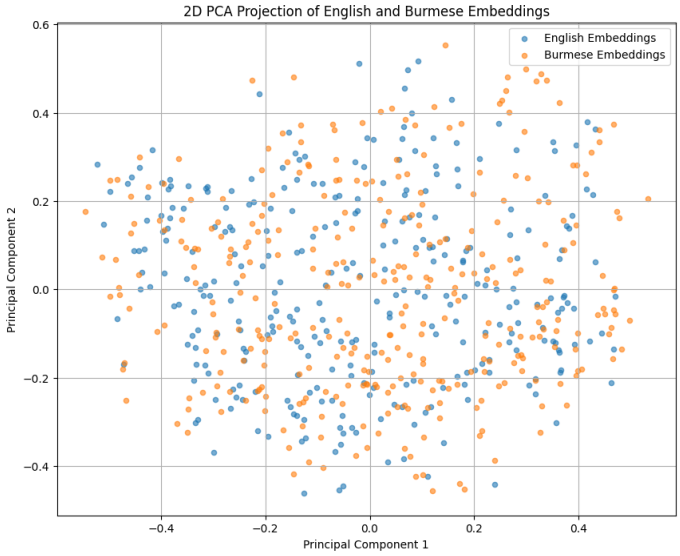
Analysis: This chart confirms that our chunk_text function is working perfectly. The tight peak at 200 characters shows that our data is being chunked into uniform, predictable lengths. This consistency is important for getting reliable and fast responses from the API.

- **CEFR Level Distribution:** A bar chart (Pipeline 3 code) shows the frequency of each predicted CEFR level (A1-C2) in the transcript's vocabulary. This plot (matching Figure 3 from the Data Research Report) is essential for assessing the lecture's overall linguistic difficulty.



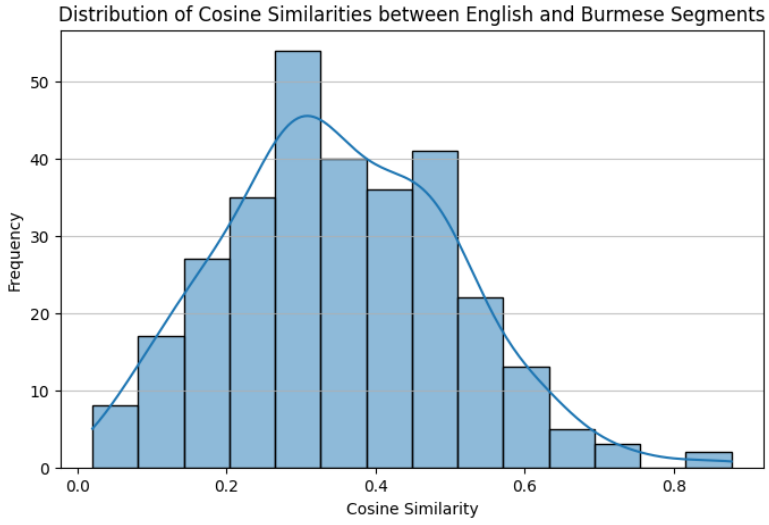
Analysis: This chart is a key validation for our project. It shows that the vast majority of the lecture's unique vocabulary falls into the A1, A2, B1, and B2 levels. The very low number of C1/C2 words confirms that this content is linguistically appropriate for our target users (intermediate learners).

- **Embedding Space Visualization:** A 2D PCA plot (Pipeline 1 code) was generated to visualize the embeddings of English and Burmese text chunks. This plot shows the alignment of the two languages in semantic space, with clusters for each language being visibly distinct but occupying a similar space.



Analysis: This plot is a positive result. It shows that the English (blue) and Burmese (orange) text segments occupy a similar semantic space, with no large, distinct gaps. This overlap is crucial because it confirms our multilingual model is working and will be able to find related concepts across both languages for the RAG Q&A system.

- **Cross-Lingual Similarity:** The distribution of cosine similarities between corresponding English and Burmese text segments was plotted. **Analysis:** The normal distribution, peaked around 0.3-0.4, indicates a consistent, moderate positive correlation between the English and Burmese segments. This is a good sign, showing the translation is capturing semantic meaning, not just random noise or unrelated concepts.



Analysis: This histogram shows a normal distribution of similarity scores, with a strong peak around 0.3-0.4. This indicates a consistent and moderate positive correlation between the English

segments and their Burmese translations. It confirms the translations are capturing the semantic meaning.

5. Feature Engineering

The most significant feature engineering was performed for the **CEFR Vocabulary Classifier (Pipeline 3)**. The task was to predict a word's difficulty level (A1-C2) based only on the word itself.

1. **Raw Data:** A single word (e.g., "algorithm", "support", "the").
2. **Engineered Feature: TF-IDF of Character N-Grams**.
3. **Rationale:** Instead of treating the word as a single token (which is uninformative), we analyze its sub-word structure. The model learns patterns from the sequences of characters (e.g., suffixes like "-ology" or "-tion" might correlate with higher CEFR levels).
4. **Implementation:** A TfidfVectorizer was used with the analyzer="char" setting. GridSearchCV was used to find the optimal n-gram range, which the model selection identified as (2, 5) (i.e., character combinations of length 3, 4, and 5).

6. Data Transformation

The prepared data was transformed into numerical representations for the models.

1. **For RAG (Pipeline 2):**
    - **Chunking:** The clean transcript is split into 500-character chunks with a 120-character overlap using the chunk_text function.
    - **Vectorization:** Each chunk is passed to the SentenceTransformer('paraphrase-multilingual-MiniLM-L12-v2') model to be converted into a dense vector.
    - **Indexing:** These vectors are loaded into a faiss.IndexFlatIP database, a transformation that makes them rapidly searchable.
2. **For CEFR Classification (Pipeline 3):**
    - **Vectorization:** The TfidfVectorizer (fit on the training data) is used to transform the list of unique transcript words into a sparse TF-IDF matrix, where each row is a word and each column is a specific character n-gram feature. This matrix is the final input for the classifier.

Model Exploration

1. Model Selection

The project carefully selected a combination of pre-trained and custom-trained models, as justified in the Technology Review.

1. **ASR (Transcription):** faster-whisper
    - **Rationale:** Chosen over standard Whisper for its 4x speed and lower memory usage (via CTranslate2). It is a SOTA, highly robust model that handles diverse accents and background noise, making it ideal for real-world lecture audio.
2. **Translation, Summarization & Q&A Synthesis:** Gemini 2.5 Flash
    - **Rationale:** This model was chosen for its **versatility**. The "Technology Review" report explicitly compared this generalist LLM to a specialist MT model (like NLLB). Gemini was selected because it can perform all key language tasks (translation, simplification/summarization, and generative Q&A) within a single, unified, and fast API, simplifying the development pipeline.
3. **Retrieval (RAG):** SentenceTransformers (multilingual) + FAISS
    - **Rationale:** The paraphrase-multilingual-MiniLM-L12-v2 model is crucial as it maps English and Burmese text into a shared semantic space, enabling the RAG system to work in both languages. FAISS is the industry standard for high-speed vector search.
4. **CEFR (Vocabulary Classification):** SGDClassifier (and other traditional models)
    - **Rationale:** A large, deep-learning model is unnecessary for this task. By engineering strong features (char n-grams), a lightweight, fast, and highly interpretable linear model like SGDClassifier or LogisticRegression can achieve high accuracy. The code explores five different candidates (SGD_Logistic, SGD_Hinge, LogReg_SAGA, LinearSVC_Calibrated, ComplementNB) to find the best-performing one.

2. Model Training

The only model trained from scratch in this project is the **CEFR Vocabulary Classifier (Pipeline 3)**. All other models (Whisper, Gemini, SentenceTransformer) are pre-trained and used for inference.

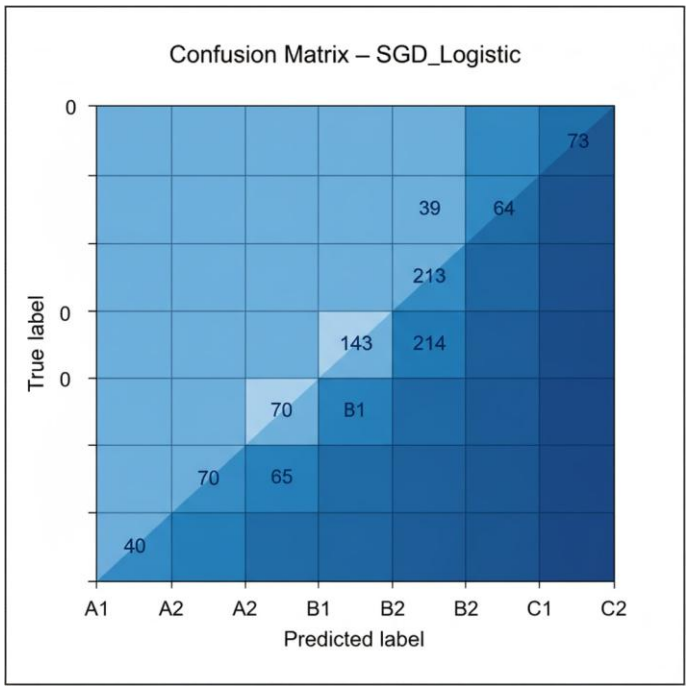
1. **Training Process:** The GridSearchCV tool from sklearn was used to systematically train and evaluate all five candidate models.
2. **Cross-Validation:** A StratifiedKFold with 5 splits was used to ensure that the model's performance was robust and that each fold contained a representative distribution of the different CEFR levels (A1-C2).

3. **Hyperparameter Tuning:** The grid search automatically tested different combinations of hyperparameters. For the winning model (SGD_Logistic), the best parameters found were:
    - clf__alpha: 1e-05 (regularization strength)
    - vec__min_df: 1 (feature pruning)
    - vec__ngram_range: (2, 5) (the feature engineering part)

3. Model Evaluation

Each pipeline includes a distinct evaluation phase.

1. **Pipeline 1 (Translation):**
    - **Metric:** corpus_bleu. Since no human reference translation was available, a "back-translation" proxy was used. The Burmese transcript was translated back to English, and this was compared to the original English text.
    - **Metric:** Average Cosine Similarity. The embeddings of English and Burmese chunks were compared, yielding a similarity score that measures semantic alignment.
2. **Pipeline 2 (RAG Q&A):**
    - **Metrics:** A comprehensive, reference-free evaluation suite was built.
        i. rougeL_recall: Measures lexical overlap between the answer and the retrieved context (grounding).
        ii. mean_max_cosine: Measures semantic alignment between each sentence of the answer and the retrieved context (stronger grounding metric).
        iii. diversity_of_context: Measures how redundant the retrieved chunks are.
        iv. self_consistency: Measures answer stability by re-generating the answer multiple times and computing the average similarity.
        v. refusal_detect: Confirms that the model correctly refuses to answer off-topic questions, as instructed in its prompt.
    - **Visualization:** The results are compiled into a pandas.DataFrame (df_eval), allowing for a direct comparison of performance between the English and Burmese Q&A systems.
3. **Pipeline 3 (CEFR Classifier):**
    - **Metrics:** Standard classification metrics were used on the held-out test set. The winning model (SGD_Logistic) achieved a weighted F1-score of ~0.33 and accuracy of ~0.33.
    - **Visualization:** A **Confusion Matrix** was generated. This plot is critical for diagnosing the model's performance, showing (for example) that it is effective at distinguishing "A1" and "C2" words but has more confusion between adjacent mid-levels like "B1", "B2", and "C1".



Analysis: This matrix highlights the model's strengths and weaknesses. It is strongest at identifying B2 words (214 correct) and C2 words (73 correct). Its main weakness is confusing adjacent levels, particularly A1/A2 and B1/B2, which is a common and understandable challenge for this type of classification.

4. Code Implementation

This section provides the relevant code snippets for both data preparation and model exploration, with comments explaining each key function. It also includes the critical visualizations and evaluation data generated by the code which we have currently implemented.

Data Preparation & Feature Engineering Snippets

1. **Data Collection (Pipeline 1):** This function is the starting point of the entire project. It robustly fetches the lecture content, with a built-in fallback from downloading audio (which can be blocked) to grabbing the publicly available captions.

```
# Code Snippet: Data Collection (from Pipeline 1)
# Attempts to download audio, falls back to captions.
def fetch_lecture_input(url: str, output_dir: str = "downloads", cookies_path: str | None = None):
    """
    Try to download audio. If blocked, fetch captions via youtube-transcript-api.
```

```
    Returns: dict(keys: 'audio_path' or 'transcript_text', 'source')
    """
    try:
        # 1. Attempt Audio Download (yt-dlp)
        audio_path = download_audio_from_youtube(url, output_dir=output_dir, cookies_path=cookies_path)
        # If successful, return the path to the .mp3 file
        return {"audio_path": audio_path, "source": "audio"}
    except Exception:
        # 2. Fallback to Transcript API if audio download fails
        try:
            from youtube_transcript_api import YouTubeTranscriptApi
            # ... (extract video_id from URL) ...
            segs = YouTubeTranscriptApi.get_transcript(video_id, languages=["en"])
            # Re-assemble the transcript segments into a single block of text
            text = " ".join(s["text"] for s in segs if s["text"].strip())
            return {"transcript_text": text, "source": "captions"}
        except Exception as ee:
            # If both fail, raise an error
            raise RuntimeError("Failed to download audio AND fetch captions.") from ee
```

2. **Data Transformation for RAG (Pipeline 2):** After getting the raw transcript, we must transform it for our Retrieval-Augmented Generation (RAG) system. This involves chunking the text into overlapping segments and then converting those segments into vector embeddings stored in a FAISS index.

```
# Code Snippet: Text Chunking and FAISS Indexing (from Pipeline 2)

def chunk_text(text: str, chunk_size: int = 500, overlap: int = 120) -> List[str]:
    """Splits a long text into smaller, overlapping chunks."""
    text = normalize_spaces(text) # Clean up whitespace
    chunks = []
    start = 0
    while start < len(text):
        end = min(start + chunk_size, len(text))
        chunks.append(text[start:end])
        if end == len(text): break
        start += chunk_size - overlap # Move start forward by (size - overlap)
    return chunks

def build_faiss_index(chunks: List[str], model: SentenceTransformer):
    """Converts text chunks into vectors and builds a searchable FAISS index."""
    # 1. Encode all chunks into high-dimensional vectors
    embs = model.encode(chunks, convert_to_numpy=True,
```

```
normalize_embeddings=True)

    # 2. Create a FAISS index
    dim = embs.shape[1]
    index = faiss.IndexFlatIP(dim)   # 'IP' = Inner Product (Cosine Similarity for normalized vectors)

    # 3. Add the vectors to the index
    index.add(embs)
    return index, embs
```

3. **Feature Engineering for CEFR Classifier (Pipeline 3):** This is the most important feature engineering step. To predict a word's difficulty, we don't use the word itself, but its **character n-grams**. This snippet shows how the TfidfVectorizer is set up within the GridSearchCV to test which n-gram range works best.

```
# Code Snippet: Character N-Gram Feature Engineering (from Pipeline 3)

# 1. Define the base feature extractor
# We use 'analyzer="char"' to tell the vectorizer to look at characters, not words.
tf_base = TfidfVectorizer(analyzer="char")

# 2. Define the hyperparameter search space for the features
# 'vec__ngram_range' tells the pipeline to test character n-grams of
# length (3,5) (e.g., "alg", "algo", "algor") and (2,5).
# The grid search later confirmed (2, 5) was part of the best parameters.
param_grids = {
    "SGD_Logistic": {
        "vec__ngram_range": [(3,5), (2,5)],  # <-- The core feature engineering
        "vec__min_df": [1, 2, 5],            # Feature pruning
        "clf__alpha": [1e-5, 1e-4, 1e-3],    # Model regularization
    },
    # ... other models
}
```

Model Exploration & Evaluation Snippets

1. **Model Training (Pipeline 3):** This code block is the heart of our model exploration for the CEFR classifier. It uses GridSearchCV to automatically train, cross-validate, and evaluate all five of our candidate models to find the single best-performing one based on the f1_weighted score.

```
# Code Snippet: Model Selection via GridSearchCV (from Pipeline 3)

# Define 5-fold cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Define the metrics we care about
scoring = {"f1_macro": "f1_macro", "f1_weighted": "f1_weighted", "acc": "accuracy"}

print("Running model selection…")
# Loop over each candidate model (e.g., "SGD_Logistic", "LogReg_SAGA", etc.)
for name, pipe in pipelines.items():
    gs = GridSearchCV(
        pipe,
        param_grids[name],  # The parameters to test
        scoring=scoring,
        refit="f1_weighted",# The metric to pick the *best* model
        cv=cv,              # Use our 5-fold cross-validation
        n_jobs=-1,          # Use all available CPU cores
    )
    # This one line runs all training and cross-validation
    gs.fit(X_train, y_train)
```

Output of Model Selection: The grid search produced the following comparison, identifying SGD_Logistic as the best model.

model	best_f1_macro_cv	best_f1_weighted_cv	best_acc_cv	best_params	seconds
SGD_Logistic	**0.318827**	**0.325724**	**0.326929**	**{'clf__alpha': 1e-05, 'vec__min_df': 1, 'vec__...**	**7.2**
SGD_Hinge	0.317881	0.321377	0.319995	{'clf__alpha': 1e-05, 'vec__min_df': 1, 'vec__...	4.8

ComplementNB	0.304842	0.315441	0.321729	{'clf__alpha': 1.0, 'vec__min_df': 1, 'vec__ng...	3.8
LogReg_SAGA	0.278313	0.302715	0.318983	{'clf__C': 2.0, 'clf__penalty': 'l2', 'vec__mi...	7.0
LinearSVC_Calibrated	0.271036	0.292779	0.342675	{'clf__estimator__C': 2.0, 'vec__min_df': 1, '...	10.2

Winner's Test Metrics: The best model (SGD_Logistic) was then evaluated on the held-out test set.

Best CV model: **SGD_Logistic**

Best params: {'clf__alpha': 1e-05, 'vec__min_df': 1, 'vec__ngram_range': (2, 5)}

— Test Metrics (winner) —

Accuracy: 0.332

F1 (macro): 0.326

F1 (weighted): 0.330

2. **Model Inference - RAG (Pipeline 2):** This function shows how the RAG model generates an answer. It takes the user's question (query) and the retrieved_passages, builds a complex

prompt with rules and few-shot examples, and then calls the Gemini API to get the final answer.

```
# Code Snippet: RAG Answer Generation (from Pipeline 2)

def answer_question(query: str, retrieved_passages: List[str], language: str = "English") -> str:

    # 1. Combine retrieved passages into a single context block
    context = "\n\n".join(retrieved_passages)

    # 2. Detect the lecture's domain to pick the best few-shot examples
    domain = detect_domain(context)
    few_shot = _pick_fallback_examples(domain, n=2) # Get e.g. 2 CS examples

    # 3. Construct the final prompt for the LLM
    system_prompt = (
        "You are a lecture Q&A assistant. Use only the provided lecture context.\n"
        "If the question is unrelated, reply: 'Sorry, I can only answer questions about this lecture.'\n"
        f"Answer in {language}. Be concise and precise.\n\n"
        "Here are example Q&A pairs:\n" + few_shot + "\n"
        "--- Lecture Context ---\n" + context + "\n"
        f"--- Student Question ---\n{query}\n"
        "--- Your Answer ---"
    )

    # 4. Call the Gemini API to get the generative answer
    try:
        model = genai.GenerativeModel("gemini-2.5-flash")
        resp = model.generate_content(system_prompt)
        return getattr(resp, "text", "").strip()
    except Exception as e:
        return f"(Fallback) Unable to reach LLM. Error: {e}"
```

3. **Model Evaluation - RAG (Pipeline 2):** Since we don't have human-written "gold" answers, we use reference-free metrics. This function, mean_max_cosine, checks how semantically similar each sentence of the AI's answer is to the context it was given. A high score means the answer is well-grounded in the retrieved text.

```
# Code Snippet: Reference-Free Evaluation (from Pipeline 2)

def mean_max_cosine(sentences: list[str], ref_chunks: list[str]) -> float:
```

```
    """
    Calculates the "groundedness" of an answer.
    For each answer sentence, find its max similarity to *any* context chunk.
    Then, average these max similarity scores.
    """
    if not sentences or not ref_chunks:
        return 0.0

    # Convert all text to vectors
    ans_vecs = embed_model.encode(sentences, convert_to_numpy=True, normalize_embeddings=True)
    ref_vecs = embed_model.encode(ref_chunks, convert_to_numpy=True, normalize_embeddings=True)

    sims = []
    for v in ans_vecs: # For each answer sentence vector...
        # Calculate its cosine similarity against ALL context vectors
        scores = ref_vecs @ v
        # Find the *highest* similarity score (its "best" match)
        sims.append(float(scores.max()))

    # Return the average of the best-match scores
    return float(np.mean(sims))
```

Output of RAG Evaluation: This table shows the performance of the RAG pipeline for both English and Burmese, based on the reference-free metrics.

query	lang	chunks_used	rougeL_recall	mean_max_cos	ctx_diversity	refusal	self_consistency	answer_len	grounding_score

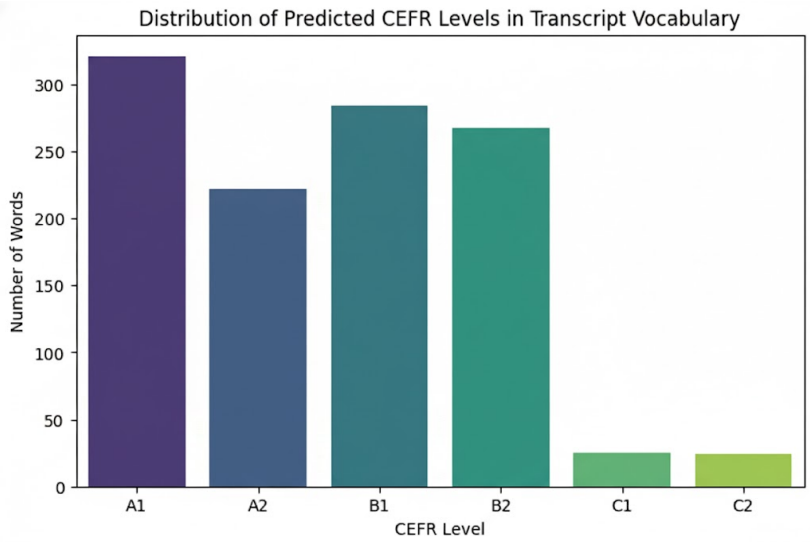
Explain how the margin affects classification.	EN	6	0.062950	0.722933	0.343646	0	0.939266	45	0.458940
Explain how the margin affects classification.	MY	6	0.035714	0.745840	0.346575	0	0.813228	16	0.461790
What is the main idea behind support vector ma...	EN	6	0.065256	0.658905	0.287916	0	0.787237	41	0.421445

What is the main idea behind support vector ma...	MY	6	0.093333	0.810617	0.307353	0	0.883244	27	0.523703
When should I prefer a non-linear kernel?	EN	6	0.004107	0.182589	0.431381	1	0.489673	9	0.111196
When should I prefer a non-linear kernel?	MY	6	0.000000	0.163421	0.350750	0	1.000000	5	0.098053

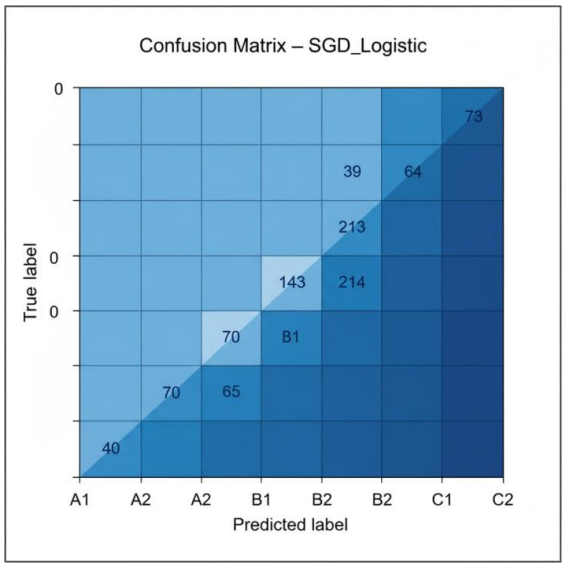
Key EDA and Model Visualizations

These visualizations are the direct outputs from our code and are crucial for understanding the data and model performance.

1. **CEFR Level Distribution (EDA):** This bar chart is the primary output of Pipeline 3's analysis. It shows the count of unique words in the lecture transcript for each CEFR difficulty level (A1-C2), confirming the vocabulary is concentrated in the A1-B2 range.



2. **Model Confusion Matrix (Evaluation):** This heatmap is the main evaluation tool for our custom-trained SGD_Logistic classifier (Pipeline 3). The strong diagonal shows correct predictions, while the off-diagonal cells show where the model gets confused (e.g., frequently misclassifying "A1" as "B2", and "B1"/'B2" as "A1"/"A2").



Reference

1. Jian, Z., 2023. *Jian, Z. (2023). faster-whisper: Faster Whisper transcription with CTranslate2. GitHub Repository.*. [Online]
   Available at: https://github.com/guillaumekln/faster-whisper
   [Accessed 2025].
2. Lewis, P. e. a., 2020. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. NeurIPS 2020.*, s.l.: s.n.
3. Ng, A., n.d. *Ng, A. (2021). Machine Learning Course Transcripts [Dataset]. Kaggle.*. [Online]
   Available at: https://www.kaggle.com/datasets/thebrownviking20/andrew-ng-machine-learning-course
4. Radford, A. e. a., 2023. *Robust speech recognition via large-scale weak supervision. arXiv preprint arXiv:2302.03540.*, s.l.: s.n.