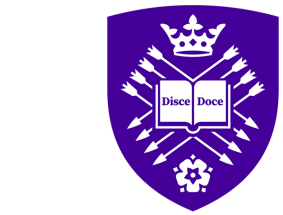

University of Sheffield

Multimodal Dense Video Captioning Using Audio, Visual, and ASR Inputs

Daniel Vousden

Supervisor: Zhixiang Chen

A report submitted in fulfilment of the requirements for the degree of BSc in Computer Science

in the

School of Computer Science

October 3, 2025

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: Daniel Vousden

Signature: DANIEL VOUSDEN

Date: 09/12/2024

Abstract

This dissertation explores the integration of audio features into dense video captioning pipelines. Audio is a modality often neglected in current models that focus primarily on visual and speech data. The project builds upon the Vid2Seq framework (Yang et al., 2023) by introducing Wav2Vec Base (Baevski et al., 2020) for audio encoding, enabling a three-stream input of vision, automatic speech recognition (ASR), and raw audio features. The aim was to improve contextual understanding and temporal alignment of generated captions. Evaluation on the YouCook2 dataset (Zhou et al., 2018) demonstrated modest gains in temporal metrics, particularly at shorter segment lengths, while maintaining comparable performance on semantic caption quality. The results support the hypothesis that incorporating audio enriches temporal segmentation and adds contextual value. Despite computational constraints and token input limits, the extended model showed promise for future multimodal captioning systems.

Contents

List of Figures

Chapter 1

Introduction

In today's world, where internet content is created and consumed on such a large scale, ensuring it is accessible to a diverse audience is paramount and remains a significant challenge. While information has become widely available, tools to ensure accessibility have lagged behind and are often seen as too time-consuming for creators to implement themselves. Dense video captioning provides a promising solution for these issues by generating descriptive text captions for the key events within videos. This idea has a broad set of applications, ranging from accessibility and education to training AI models and multimedia content indexing.

Due to multi-modal dense captioning being a newer research endeavor, most approaches to the problem are heavily focused on visual feature analysis. Current research approaches engineer their models to focus on improving temporal analysis, such as determining time boundaries for events effectively and processing video frames to identify key features. While these components are very important for dense caption generation and progress has been made, the role that audio plays in this process have been under explored. In some cases, audio analysis is not treated as a priority or is ignored altogether, limiting the richness and accuracy an effective set of audio features could bring to the caption generation process. This oversight can have a noticeable impact in situations where sound plays a crucial role, such as telling apart spoken dialogue, picking up background noises, or understanding emotional cues.

Bringing richer audio analysis and feature generation into a dense captioning pipeline presents unique challenges but also opens up many possibilities. It's not easy to separate meaningful sounds from background noise, align them seamlessly with visual events, or ensure their semantic value is evenly captured across multi-modal inputs. But tackling these challenges could bring about a significant improvement in the quality and depth of dense captions, opening the door to more impactful and inclusive applications.

1.1 Aims and Objectives

During the time spent on this project I aim to address the under-utilization of audio features in dense video captioning. By integrating more thorough audio feature extraction methods,

the goal is to enhance the descriptive and contextual potential of generated dense captions. The key objectives include the following:

- Explore and understand existing methods for multi-modal dense captioning.
- Identify and address gaps in audio integration into these existing solutions.
- Develop an approach to utilize enhanced audio analysis into the dense captioning pipeline.

1.2 Overview of the Report

The subsequent chapters of this report delve into a comprehensive examination of dense video captioning. Chapter 2 presents a literature survey, discussing the evolution of dense captioning and analyzing current approaches and their limitations. Chapter 3 outlines the project's requirements, component analysis, and a feasibility study. Chapter 4 tracks the progress made and outlines future plans. Finally, Chapter 5 concludes with a discussion on the findings and implications of the research. Each chapter builds on the insights developed in this analysis to propose and validate enhancements in multi-modal dense captioning.

Chapter 2

Literature Survey

2.1 Introduction

The purpose of this literature review is to explore the current research within the field of dense video captioning, a research field that has grown more relevant due to advancements in machine learning and multi-media analysis. The field originally developed from the interest of producing single sentence captions for static visual frames and has evolved into describing key events within video clips. The evolution of this process has required the implementations of temporal and multi-modal features aided by advancements in deep learning leading to more complex but effective solutions. Dense video captioning, which generates contextual and descriptive text for bounded events, has seen a large focus in the temporal and visual aspects of the problem yielding significant progress. However, the impact that audio features could have on the resulting caption generation has been under explored, such as music, speech and background noise. These extra considerations in the audio processing field could potentially provide emotional context and richness to captions that can't be captured effectively otherwise.

The review examines the current state of the art research within dense captioning, focusing on the model's architecture, specific module implementations, the results obtained and the inputs they used to obtain these results. By analyzing these existing models and identifying gaps in the existing research, the review establishes a foundation for implementing audio features into the captioning pipeline. The chapter is constructed as follows: Section 2.2 explores the background on the evolution of dense video captioning; Section 2.3 reviews the existing approaches to feature representation, temporal event localization, and multi model integration; and section 2.4 critiques the gaps in current methodologies, linking them to the goals of this project.

2.2 Evolution of Dense Captioning

The topic of dense captioning has long been a constantly evolving topic of research and has evolved from early efforts in single-image captioning to trying to fully capture the contents of

a dynamic video. Initial work (Vinyals et al., 2015) , used neural networks to caption static images, this demonstrated the feasibility of being able to produce text outputs from visual inputs. This research and methodology created a strong foundation for video captioning, where models could begin summarizing video clips into single captions (Venugopalan et al., 2015).

This advancement was great for capturing high-level summaries, but it lacked in capturing the dynamic, event driven structure of video clips. This limitation led to the introduction of dense video captioning, introduced by Krishna et al. (2017) through the ActivityNet Dense-Captioning Dataset (Caba Heilbron et al., 2015). Their framework implemented a focus on temporal event proposals and combined this with caption generation, marking a shift towards event detection in dense captioning solutions.

As the field developed, more deep learning techniques were considered, such as temporal convolutional networks (Xu et al., 2019) and transformer-based models (Wang et al., 2021), to improve event boundary detection and the generated caption quality. While these advancements made huge strides in dense captioning, they predominantly focused on visual inputs only, overlooking other data inputs such as audio despite its potential to integrate more emotional and contextual depth into the captions.

2.3 Approaches to Dense Captioning

2.3.1 PDVC

This section comprehensively applies the methodologies and findings discussed in Wang et al. (2021) to explore the integration of feature extraction and caption generation in dense video captioning systems. The approaches outlined by Wang et al. (2021) form the foundation for all discussions and analyses presented throughout this section.

The Parallel Decoding Video Captioning (PDVC) model introduces an end-to-end approach to the task of dense video captioning as shown in Fig. 2.1 (Wang et al., 2021), viewing the process as a prediction task. By using associations at the feature level, the model is able to integrate the generation of event boundaries and captions in the same feature space causing them to share the same features. A key innovation of this model was a development on the boundary-based event localization method and it include using a novel event counter to estimate the number of events in a video. This enhances caption readability by preventing unrealistic event predictions. Through the use of reference points to focus attention on specific parts of frames and employing one-to-one matching between intermediate feature vectors and target event instances. PDVC achieves state of the art performance when compared to the traditional two step methods while reducing reliance of extensive fine-tuning.

Preliminary: Deformable Transformer

Due to the PDVC model wanting to integrate feature extraction, event localisation, and caption generation into a unified and streamlined framework, the deformable transformer is

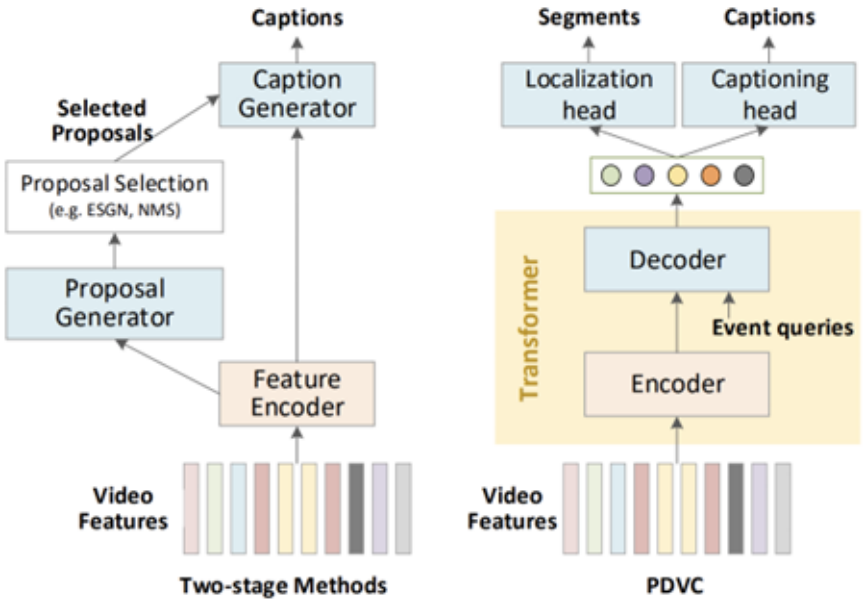

Figure 2.1: PDVC's architecture (Wang et al., 2021).

repeatedly used throughout the model as shown in Fig. 2.2 (Wang et al., 2021).

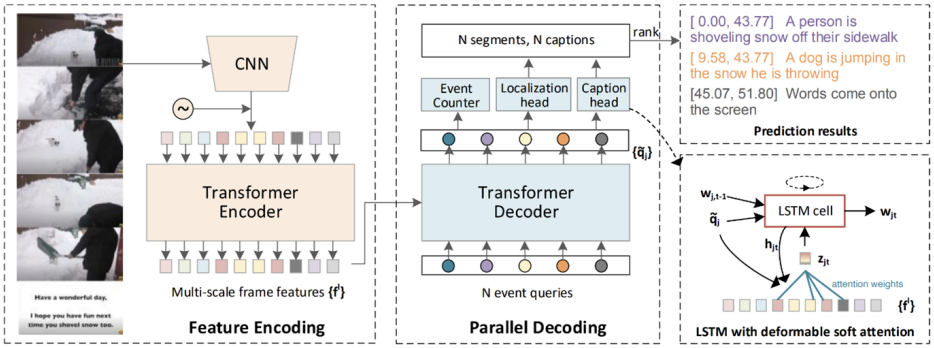

Figure 2.2: PDVC's architecture detailing the modules (Wang et al., 2021).

The deformable transformer within the PDVC framework uses a encoder-decoder architecture based on multi-scale deformable attention. This approach addresses the slow convergence issues associated with self attention mechanisms in traditional transformers when processing image feature maps. It achieves this by focusing on a sparse set of sampling points around reference points in the frames, which significantly reduces computational overhead compared to analyzing the whole frame.

Given multi-scale feature maps:

$$X = \{X_l\}_{l=1}^{L}, \quad X_l \in \mathbb{R}^{C \times H \times W},$$

where q_j is a query element, and $P_j \in [0,1]^2$ is a normalized reference point. The multi-scale deformable attention mechanism (MSDAtt) computes a context vector by the weighted sum of $K \times L$ sampling points across feature maps at L scales:

$$\text{MSDAtt}(q_j, P_j, X) = \sum_{l=1}^{L} \sum_{k=1}^{K} A_{jlk} W X^l_{\hat{P}_{jlk}},$$

where:

$$\hat{P}_{jlk} = \phi_l(P_j) + \Delta P_{jlk}.$$

Here:

- $\hat{P}_{jlk}$ and A_{jlk} represent the position and attention weight of the k-th sampled key at the l-th scale for the j-th query element, respectively.
- W is the projection matrix for key elements.
- ϕ_l projects normalized reference points into the feature map at the l-th level.
- ΔP_{jlk} are the sampling offsets relative to $\phi_l(P_j)$.

Both A_{jlk} and ΔP_{jlk} are obtained through linear projection applied to the query elements.

Additionally, the original MSDAtt incorporates a multi-head attention mechanism, allowing the model to better capture diverse relationships across feature maps.

This deformable attention mechanism allows for efficient and accurate modeling of inter-frame and inter-event relationships. It ensures the PDVC model can achieve robust feature extraction, event localization and caption generation.

Feature Encoding

The first step in the model is to extract feature representations of the frames input, as shown in the first box of Fig 2.2 (Wang et al., 2021). The feature extraction method chosen is to adopt a pre-trained action recognition network to capture rich spatio temporal feature within the video. The frame level features are then extracted, ensuring that important visual elements are represented. In order to enable batch processing, the temporal dimension of the extracted feature maps is scaled to a fixed size. Multi scale features are then utilized by applying L temporal convolutional layers with a stride of 2 and kernel size of 3. The implementation of this approach allows for features across multiple resolutions to be captured, improving the models ability to identify patterns on different temporal scales. These multi scale frame features are then fed into the deformable transformer encoder, which refines the features by extracting frame-by-frame relationships across the multiple scales.

Parallel Decoding

The Parallel Decoding mechanism in the PDVC framework consists of a deformable transformer decoder and three parallel heads, each addressing a specific task: caption generation, event

boundary prediction, and event count estimation. The captioning head generates descriptive captions for events, the localization head predicts event boundaries along with their confidence scores, and the event counter predicts the appropriate number of events for the input video.

The decoder is designed to directly query event-level features from the frame-level features, conditioned on N learnable embeddings $\{\mathbf{q}_j\}_{j=1}^{N}$ and their corresponding scalar reference points $\mathbf{p}_j$. Each reference point $\mathbf{p}_j$ is predicted using a linear projection with a sigmoid activation applied to the query embedding $\mathbf{q}_j$. The event queries and reference points serve as initial approximations of the events' features and locations. These approximations are iteratively refined at each decoding layer of the transformer.

After the iterative refinement process, the output query features and the reference points are denoted as $\mathbf{q}_j, \mathbf{p}_j$, respectively, capturing the updated event features and their localized positions.

Localization Head

The localization head is a sub part of the parallel decoding process taking place after the transformer decoder as shown in Fig. 2.2 (Wang et al., 2021). The goal of this head is to generate a confidence score for each potential event that has been identified thus far, by performing box prediction and binary classification for each event query. The box prediction aspect aims to predict the 2D relative off-sets (center and length) of the ground truth segment with respect to the reference point, which is the original approximated event point. This process allows for event boundaries to be refined during training and inference. The binary aspect of this process then has the responsibility of calculating the foreground confidence of each query produced. Both of these processes are implemented using a multi-layer perceptron.

At the end of this process we obtain a set of tuples:

$$\{t_j^s, t_j^e, c_j^{\text{loc}}\}_{j=1}^{N}$$

to represent the detected events, where c_j^{loc} is the localization confidence of the event query q_j.

Captioning Head

The captioning head is another sub part of the parallel decoding process. The goal of this head is converts event-level features which are produced by the encoder and refined by the decoder into coherent, descriptive captions for each event. The captions are generated one caption at a time, ensuring contextual coherence as the previously generated tokens can be used in the generation of the next. The model introduces two versions of this captioning head, one being the standard version and the other being a lightweight version to conserve computational resources.

In the standard version the decoder takes event queries q_j, reference points p_j, and the previous hidden state h_{t-1}. The dynamic context vector c_t is computed using the process deformable soft attention and these inputs. The deformable soft attention module is then

used to calculate attention weights dynamically around the reference points p_j, focusing on regions of interest.

The sampling points $\{p_{jlk}\}$ are generated as:

$$p_{jlk} = \phi_l(p_j) + \Delta p_{jlk},$$

where:

- $\phi_l(p_j)$: Projects the normalized reference points p_j into the feature map at scale l,
- Δp_{jlk}: Represents learned offsets relative to $\phi_l(p_j)$.

Then for each sampling point, the attention weights are calculated as:

$$A_{jlk} = \text{Softmax}\left(W_{\text{att}} \cdot [h_{t-1}, q_j, p_{jlk}]\right),$$

where:

- W_{att}: Attention weight projection matrix.

The sampling points $\{p_{jlk}\}$ are then used to locate the features X_{jlk} to compute the context vector:

$$c_t = \sum_{l=1}^{L} \sum_{k=1}^{K} A_{jlk} \cdot X_{jlk}$$

The hidden state layer is then updated:

$$h_t = \text{LSTM}([c_t, q_j, w_{t-1}], h_{t-1})$$

where:

- c_t: The context vector
- q_j: Event queries
- w_{t-1}: The previous word

Now that the hidden state has been update it can be used to predict the next word:

$$P(w_t \mid h_t) = \text{Softmax}(W_{\text{out}} h_t + b_{\text{out}})$$

where:

- W_{out}: Output weight matrix,
- b_{out}: Bias vector.

In contrast the implementation of the lightweight head is a far more streamlined version of the standard head. This allows for a result to be produced using far less computational resources even though the result deteriorates a considerable amount. At each time step t, the LSTM receives the event level feature q_j from the deformable transformer decoder. This is then used directly to update the hidden state h_t:

$$h_t = \text{LSTM}(h_{t-1}, q_j),$$

where:

- h_{t-1}: The previous hidden state.

The new hidden state can then be used to predict the next word using a fully connected layer followed by a softmax:

$$P(w_t \mid h_t) = \text{Softmax}(W_{\text{out}} h_t + b_{\text{out}}),$$

where:

- W_{out}: Weights of the output layer.
- b_{out}: Biases of the output layer.

Event Counter

The final part of the parallel decoding module is the event counter. The purpose of this counter is to ensure an appropriate number of events are identified. This is an innovation this model brought forward and alleviates many issues with previous models such as too many events causing repeated captions or too few events causing missing parts of the story. The counters detects the number of events by using a max-pooling layer and a fully connected layer with soft-max activation. It compresses the information form the event queries into a global feature vector which is used to predict a fixed size probability distribution where each value in the distribution represents the likelihood of a specific event count. During inference the predicted event number is obtained by:

$$N_{\text{set}} = \arg\max(\text{r}_{\text{len}}).$$

The final output then selects the top N_{set} events with accurate boundaries and good queries from N event queries. These are determined by a confidence score.

$$c_j = c_j^{\text{loc}} + \frac{\mu}{M_j \cdot \gamma} \sum_{t=1}^{M_j} \log(c_{jt}^{\text{cap}})$$

- Where c_{jt}^{cap} is the probability of the generated word.
- The modulation factor μ is added to rectify the influence of caption length.

- γ is the balance factor.

μ is added because it has been observed that the average confidence is not a convincing measurement of sentence level confidence since the captioning head often overestimates confidence for shorter sentences.

Training and Loss

During training, the model predicts a set of N events, each with a location and caption. To align the predicted events with ground truth events, the Hungarian algorithm is used for optimal bipartite matching. The matching cost is defined as:

$$C = \alpha_{\text{giou}} L_{\text{giou}} + \alpha_{\text{cls}} L_{\text{cls}}$$

where:

- L_{giou} represents the generalized Intersection over Union (gIOU) between predicted and ground-truth temporal segments.
- L_{cls} represents the focal loss between the predicted classification score and the ground-truth label.
- α_{giou} and α_{cls} are corresponding weights.

After matching the set prediction loss can be calculated as the weighted sum of the two loss components:

$$L = \beta_{\text{giou}} L_{\text{giou}} + \beta_{\text{cls}} L_{\text{cls}} + \beta_{\text{ec}} L_{\text{ec}} + \beta_{\text{cap}} L_{\text{cap}}$$

where:

- L_{giou}: Generalized IOU loss.
- L_{cls}: Classification loss.
- L_{ec}: Counting loss, which uses cross-entropy between predicted and true count distributions.
- L_{cap}: Caption loss, measuring cross-entropy between predicted word probabilities and the ground-truth caption, normalized by caption length.
- $\beta_{\text{giou}}, \beta_{\text{cls}}, \beta_{\text{ec}}, \beta_{\text{cap}}$: Corresponding weights.

The final loss is the sum of the set prediction losses across all transformer layers, ensuring comprehensive training across different stages of the model.

Results

In event localization PDVC outperforms MFT and matches SDVC without relying on hand tuned proposal modules. The use of parallel set prediction improves efficiency. This is even more true in longer videos. In dense captioning PVDC achieves state of the art results with +22.22% BLEU4 and +75.87% CIDEr. PDVC performs faster than 2 stage methods with PDVC Light: 0.09s/video and PDVC: 0.16s/video. It was also observed that captioning supervision improves proposal descriptiveness and boosts caption quality, deformable components proved vital for localization, optimized query number balance precision and recall, and a higher modulation factor yield better results.

2.3.2 Vid2Seq

This section comprehensively applies the methodologies and findings discussed in Yang et al. (2023) to explore the integration of feature extraction and caption generation in dense video captioning systems. The approaches outlined by Yang et al. (2023) form the foundation for all discussions and analyses presented throughout this section.

Vid2Seq is a transformer-based dense captioning framework that addresses event localization and caption generation in a single, end-to-end process. It achieves this by introducing time tokens that encode temporal dependencies within the features, dynamically aligning them with timelines. This design eliminates any reliance on handcrafted components, while achieving state of the art performance.

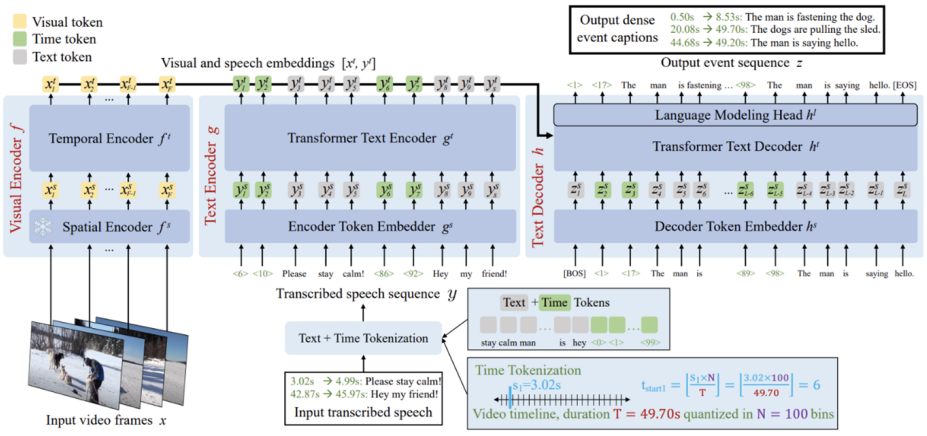

Figure 2.3: Vid2Seq's architecture detailing the modules (Yang et al., 2023).

Input Preparation

The first step in this model is to segment the inputs to give them a temporal structure. The video frames $x \in \mathbb{R}^{F \times H \times W \times C}$ (with F frames) are split into N segments and the transcribed

speech $y \in \{1, \ldots, V + N\}^S$ (with S tokens) is split into phrases based on punctuation. The speech is tokenized and aligned with timestamps in the video to provide temporal context.

Time Tokenization

Both the video and speech sequences are then augmented with time tokens. These time tokens are used to identify N evenly spaced intervals in the video input, and the time tokens t_n are appended. This ensures alignment between the video frames and speech text by anchoring them to a given point in the video.

For an event z_k, the sequence is represented as:

$$z_k = \{t_{\text{start}_k}, t_{\text{end}_k}, z_{k1}, \ldots, z_{kl_k}\},$$

where t_{start_k} and t_{end_k} represent the start and end times, and $\{z_{k1}, \ldots, z_{kl_k}\}$ describe the event.

Sequence Construction

All events are concatenated, and special tokens `BOS` (Beginning of Sequence) and `EOS` (End of Sequence) are appended:

$$z = \{\texttt{BOS}, t_{\text{start}_1}, t_{\text{end}_1}, z_{11}, \ldots, \texttt{EOS}\}.$$

Feature Extraction

The features for the visual and speech data are extracted using their respective.

Visual Encoder

Spatial Feature Extraction: The visual encoder applies a spatial backbone (CLIP ViT-L/14) f_s to extract the frame-level features:

$$x_s = f_s(x) \in \mathbb{R}^{F \times d},$$

where d is the embedding dimension.

Temporal Feature Encoding: Temporal relationships are extracted using a temporal transformer f_t, with positional embeddings x_p:

$$x_t = f_t(x_s + x_p) \in \mathbb{R}^{F \times d}.$$

Text Encoder

Token Embedding: The transcribed speech sequence y is embedded using g_s:

$$y_s = g_s(y) \in \mathbb{R}^{S \times d}.$$

Textual Contextualization: Relationships within speech tokens are modeled using a text transformer g_t:

$$y_t = g_t(y_s) \in \mathbb{R}^{S \times d}.$$

Multi-Modal Fusion

The visual tokens x_t and text tokens y_t are concatenated into a unified representation:

$$e = [x_t; y_t] \in \mathbb{R}^{(F+S) \times d}.$$

This combined representation enables the decoder to process both modalities together.

Text Decoder

Token Input: At step k, the decoder takes the previously generated tokens $z_{\leq k}$ and the combined embeddings e to predict the next token:

$$z_{tk} = h_t(h_s(z_{<k}), e) \in \mathbb{R}^d.$$

Language Modeling: The probability of the next token z_{k+1} is calculated as:

$$P(z_{k+1}|z_{\leq k}, e) = \text{Softmax}(W_o z_{tk} + b_o),$$

where W_o and b_o are the weights and biases of the output layer.

Training and Loss

The training for Vid2Seq leverages large amounts of unlabeled narrated videos and uses transcribed speech on corresponding time stamps. It was also found that training this model on longer videos increased its effectiveness. The loss for this model is a measure of how well the model is predicting the next token in the sequence based on its current inputs:

$$\mathcal{L}_\theta(x, y, z) = -\frac{1}{\sum_{k=1}^{L-1} w_k} \sum_{k=1}^{L-1} w_k \log p_\theta(z_{k+1} \mid x, y, z_{1:k}),$$

where:

- $\mathcal{L}_\theta(x, y, z)$: Loss function to be minimized during training. Measures the effectiveness of the model to predict the next token in the sequence based on current inputs.
- x: Represents the visual inputs.
- y: Represents the encoder text sequence, which is the tokenized transcribed speech.
- z: Represents the decoder target sequence.

- $\sum_{k=1}^{L-1} w_k$: Represents the sum of weights across all tokens in the sequence from token 1 to $L-1$, where L is the length of the target decoder sequence.
- w_k: The weight of the k-th token in the sequence (in practice, this is typically 1 for equal weighting).
- $p_\theta(z_{k+1} \mid x, y, z_{1:k})$: The conditional probability of predicting the next token z_{k+1} given the visual inputs x, the encoder text sequence y, and the previously generated tokens $z_{1:k}$. θ refers to the trainable parameters of the model.
- $\log p_\theta$: The log probability of the model's predicted output, which helps convert the probability into a format suitable for loss computation.

The training process has two objectives, the first being the generative objective. This is where the transcribed speech is used as a pseudo-supervisory for teaching the decoder. The decoder has to predict a sequence of events given image inputs only as using the text here would only create text related shortcuts. The second objective is the de-noising objective. Due to the text not being used during training the text encoder is not trained, however this is necessary when being used. The objective is used to align the text and visual encoder by using text were random spans of the text are replaced with tokens. These tokens mask the text and the decoder has to attempt to guess them.

Results

Using untrimmed videos and integrating boundaries from transcribed speech via time tokens leads to better results than using trimmed videos, visual inputs only benefits from the generative objective showing performance improvements and adding the de-noising methods improve performance even more. It was found that larger datasets provide better results and the model that performs both event boundary prediction and caption generation performs better than the model that just predicts event boundaries. The model achieves state of the art results for dense video captioning on YouCook2 (Zhou et al., 2018). ViTT (Radford et al., 2021) and ActivityNet (Caba Heilbron et al., 2015). However it does under-perform on ActivityNet (Caba Heilbron et al., 2015) compared to more specialized event boundary detectors.

2.3.3 X-VILA

This section comprehensively applies the methodologies and findings discussed in Ye et al. (2023) to explore the integration of feature extraction and caption generation in dense video captioning systems. The approaches outlined by Ye et al. (2023) form the foundation for all discussions and analyses presented throughout this section.

X-VILA focuses on enabling seamless multi-modality conversions and content generation. While this model is designed to extend large language models instead of producing dense captions, it produces some ideas that could be transferrable to solving multi-modal dense

captioning. An example of this is how it projects all of the data modalities into the same feature space for processing allowing them to interact. The structure adopted consisted of modal specific encoders into a projection layer. All the modalities projected features are processed by the LLM and the resulting output is projected back into its own feature space and decoded. The image information is also separately processed by the Visual Embedding Highway to retain visual features.

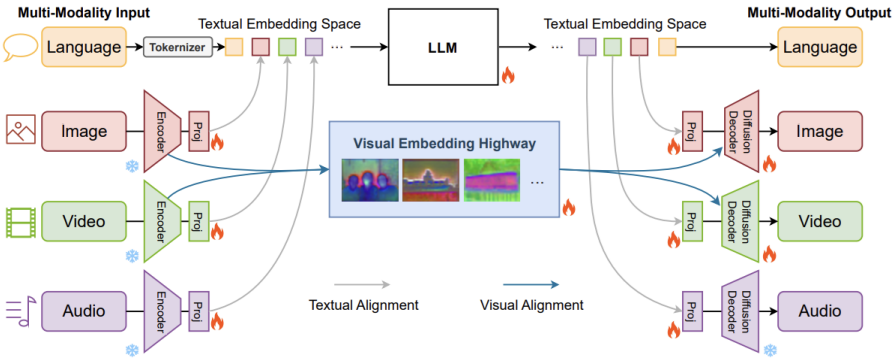

Figure 2.4: X-VILA Architecture (Ye et al., 2023).

Modality Specific Encoders

Each input modality has their own encoder which are pretrained for feature extraction. The extracted features are then projected into the textual embedding space using the modality specific linear layer. These projected features are represented as embedding sequences:

$$S_{\text{in}} = \left\{ P_m^{\text{in}} \left(\text{Enc}_m(X_m) \right) \right\}$$

For each modality m, the input X_m is:

1. Processed by the modality-specific encoder Enc_m.
2. Projected into the LLM's embedding space using the projection layer P_m^{in}.

The result is a set of embeddings S_{in} unified in the textual embedding space.

Modality Specific Decoders

After being processed by the LLM using Vicuna7B-1.5. The outputs then have to be converted into different modalities for decoding. This is achieved by adopting generation tokens which are learned and output by the LLM along with the content. The subset of the output for a given modality is then extracted (this is then called the generation embedding sequence) and trainable transformer layers are used to project the generation embedding sequence into the modalities feature space. The output of this projection layer is called the textual

controller embedding and serves as a control signal for the decoder. The signal guides the modality-specific decoder via cross-attention and it generates outputs on the embeddings aligned with its feature space.

$$E_m^{\text{text}} = P_m^{\text{out}}(S_m^{\text{gen}})$$

- P_m^{out}: Transformer layers that project the generation embeddings.
- S_m^{gen}: Modality-specific subset of the output embeddings.
- E_m^{text}: Resulting embedding controlling the decoder for modality m.

Visual Embedding Highway

This module of X-VILA takes place alongside the LLM for visual based inputs such as images and videos to retain visual features as they deteriorate when processed in text form. The implementation of this module is due to text having a one to many correspondence, meaning one word could have many images leading to images having vague text descriptions. The visual embedding highway bridges the gap between the encoders and decoders to reduce information loss during high dimensional visual to textual embedding projection.The ImageBinds visual encoder utilizes layer-wise feature maps to extract and analyze image and video input data effectively. These feature maps are then integrated into a visual highway embedding, known as Evis, which maintains the original spatial dimensions (height x width) while encoding them into a specified vector size. ImageBinds is a multi-modal learning framework designed to enhance feature extraction across different visual modalities.

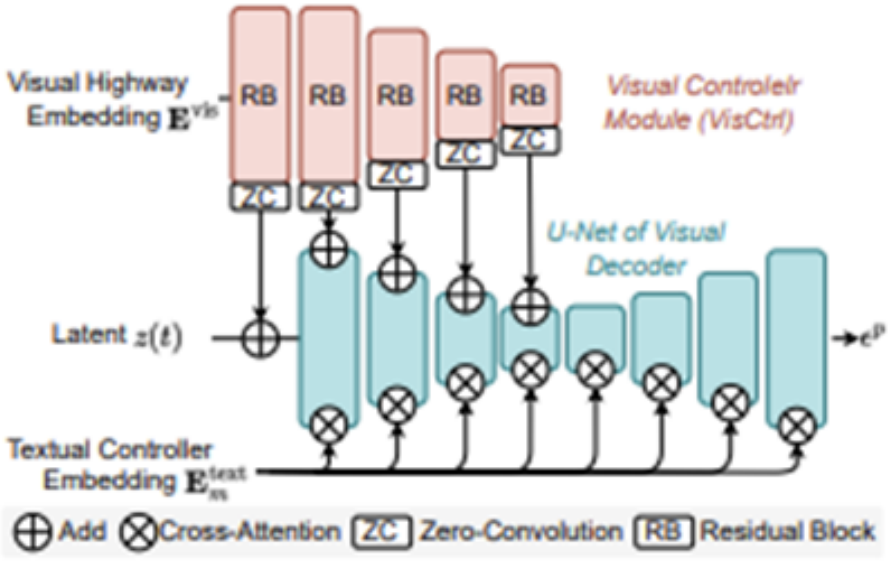

Figure 2.5: Visual Embedding Highway (Ye et al., 2023).

VisCtr

As shown in Fig. 2.5 (Ye et al., 2023), once the Evis have been produced they are processed with a lightweight module called VisCtrl. This process has four stages, each with two residual convolutional blocks. These blocks consist of a block to align their spacial resolutions with the U-Net encoder and a convolutional block initialized to 0 to generate output control signals. The control signals produced are used in the corresponding decoding stages in the U-Net decoder.

There is a conditioning rate defined during this process which is used to control the proportion of reverse steps in the U-Net influenced by visual features. This means that it will determine how much of an impact the visual features should have and how much impact the prompt should have. This process allows detailed visual context to be captured and then ensuring the visuals align with the textual context.

Training and Loss

X-VILA is trained in 3 phases which are the data-effective alignment phase, the interleaved multi-modality pre-training phase and the X-to-X cross-modality instruction tuning phase. The first phase aligns the multi-modality encoders with the LLM inputs and multi-modality decoders with LLM outputs. The second phase utilizes interleaved instruction data across modalities to enhance in-context learning performance. Finally the third phase includes a two-step alignment process (textual and visual alignment) to ensure comprehensive and accurate correspondence between input and output modalities. In the X-VILA model there are 5 loss functions used, the first being alignment loss. This loss ensures the alignment of the encoders with the LLM's textual embedding space. This is achieved by minimizing the reconstruction error between the projected encoder outputs and the LLM-processed embeddings. The second type is generation loss, which is applied to the outputs of the decoders and measures the reconstruction quality compared to the original input. Captioning loss is the third type and uses a cross-entropy loss for the generation tasks. As the LLM generates a word-by-word sequence, the loss is calculated based on the difference between the generated and ground truth captions. The fourth is regularization loss, which regularizes the alignment process to ensure the unified embedding space is we distributed and avoids over-fitting one modality. Finally the fifth type is attention loss, this optimizes the cross attention mechanism by encouraging the model to prioritize meaningful regions of input data during decoding. The overall training loss:

$$L = \lambda_1 L_{\text{alignment}} + \lambda_2 L_{\text{generation}} + \lambda_3 L_{\text{captioning}} + \lambda_4 L_{\text{regularization}}$$

where:

- $\lambda_1, \lambda_2, \lambda_3, \lambda_4$: Weights assigned to balance the contributions of each loss function.

Results

They found the visual highway improved outputs, especially for visual-visual. Video inputs also benefited from the visual highway and the model could perform novel tasks (e.g. audio to image) without direct training for that task. The model handles multiple inputs well, generating consistent outputs. X-VILA outperformed Next-GPT at X-to-X tasks with and without the visual highway.

2.3.4 Limitations of Current Approaches

PDVC

- PDVC's event counter can still lead to overestimated event numbers even though it is an improvement to previous methods (Wang et al., 2021).
- There is no consideration for multiple modalities which can lead to reduced caption accuracy in terms of context and descriptive power (Wang et al., 2021).
- Forcing event boundary prediction and caption generation to share the same features isn't always optimal (Wang et al., 2021).
- Despite major improvements performance still heavily relies on the quality of ground truth values for training (Wang et al., 2021).

Vid2Seq

- Sometimes due to the captions not perfectly aligning with the visual data, which can affect the quality of the captions produced as the captions and visual data are not temporarily aligned (Yang et al., 2023).
- The reliance on transcribed text for training may lead to weak supervision issues. Especially if the dataset contains noisy data (Yang et al., 2023).
- Using time tokenization for producing boundaries can become imprecise if there are no distinct boundaries in the text (Yang et al., 2023).
- Audio data from the videos is not considered potentially missing out on extra context and descriptive power for the generated captions (Yang et al., 2023).

X-VILA

- Although X-VILA attempts to preserve visual features, it can still struggle with more complex inputs (Ye et al., 2023).
- Due to the complex integration of modalities, outputs aren't always consistent (Ye et al., 2023).

- Using time tokenization for producing boundaries can become imprecise if there are no distinct boundaries in the text (Ye et al., 2023).
- The models outputs are highly dependent on the training data quality. Especially for new cross modality tasks (Ye et al., 2023).

A limitation shared across the three models is the lack of raw audio integrated. PDVC's joint optimization of proposal and captioning modules offers strong alignment but sacrifices flexibility for modality expansion. Vid2Seq introduces temporal tokenisation through speech, but this doesn't consider the impact of transcription quality and token limits. X-VILA demonstrates generalisable multimodal learning, but isn't tailored for the task of dense captioning. While each model advances a key area of the problem, none fully capture the potential of audio data. This indicate a big research gap audio-rich dense captioning.

Few works attempt end-to-end dense captioning that leverages ASR, visual, and non-speech audio inputs. Alternatives such as audio-scene analysis or emotion-aware embeddings are largely absent. Representing possible directions for future exploration in multi-modal captioning research.

Conclusion

As seen each model has its share of unique drawbacks and limitations. However a consistent one for the dense captioning models are their lack of consideration for audio features which could enhance the emotion, context and descriptiveness of the generated captions. This limitation directly correlates with my goals of integrating audio feature analysis into a dense captioning framework.

Chapter 3

Requirements and Analysis

3.1 Aims and Objectives

3.1.1 Aims

- Improve the accuracy and contextual relevance of video caption by using audio cues and analysis into the captioning pipeline.
- Develop a dense captioning pipeline that effectively combines audio and visual data to generate dense captions.

3.1.2 Objectives

- Extract relevant audio features from videos, such as speech and background audio.
- Ensure precise synchronization of the audio and visual features to enhance the contextual understanding of video segments.
- Adapt an existing model to use audio features as well as visual.
- Measure the quality of the captions generated by my model.
- Compare my results to models that don't use audio inputs.

3.2 Requirements Specification

3.2.1 Functional Requirements

- The framework must identify and process video, audio and speech inputs.
- The extracted features must be processed to reflect an increase in context, emotion and descriptiveness in the captions produced.
- The input data types must be temporally synchronized to ensure the accuracy of generated captions.

- The system can handle multiple audio formats as inputs.
- The user should be able to interact with the system by uploading just a video.

3.2.2 Non-Functional Requirements

- The system should perform consistently for differing visual and audio conditions.
- The system should maintain performance when dealing with noisy data.
- The system should have error handling to gracefully notify if a file entered is invalid.
- The system should aim for consistent speed performance across varying inputs.

3.3 Component Analysis

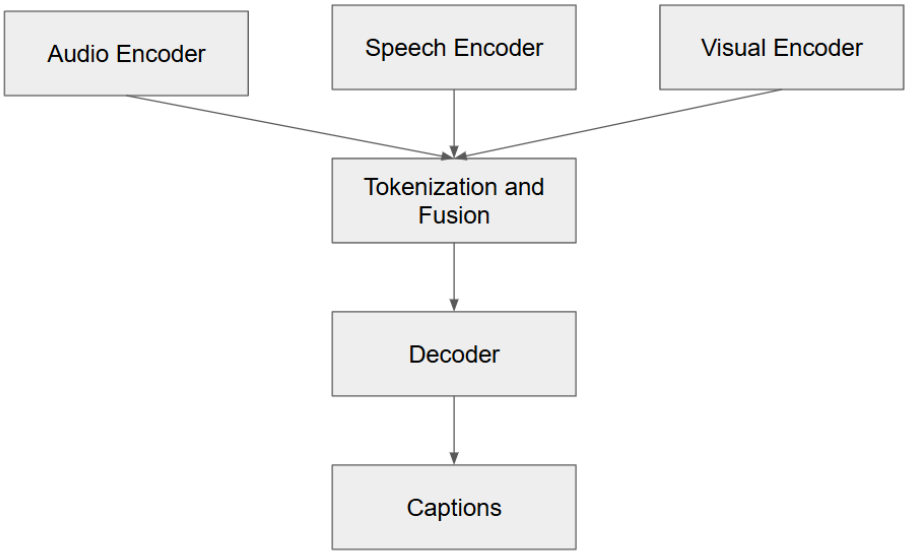

Figure 3.1: High level representation of my framework.

3.3.1 Visual Encoder

As shown in Fig. 3.1, the framework will make use of a visual encoder to extract spatial and temporal features from video frames. This will help the system understand actions, objects and scene transitions.

3.3.2 Speech Encoder

As shown in Fig. 3.1, the framework will make use of a speech encoder to extract spoken dialogue from the video.

3.3.3 Audio Encoder

As shown in Fig. 3.1, the framework will make use of an audio encoder to extract non-spoken audio from the video. This includes music, ambient noise, and sound effects. These features will contribute to identifying emotional and contextual cues.

3.3.4 Tokenization and Fusion

The output features from each encoder are converted into a tokenized form, preparing them for integration. These token streams are aligned temporally and passed to the decoder.

3.3.5 Decoder

The decoder processes the tokens and generates event-level captions. The text segments output will describe temporally aligned segments of the video.

3.4 Feasibility Study

3.4.1 Available Technologies

The project will make use of advanced technologies such as deep learning frameworks (PyTorch) for processing audio and video data. These are able to handle data processing and the task of combining multiple pre-trained models which is essential for this project.

3.4.2 Technical Resources

The implementation of this project will require access to computational resources. Due to the use of pre-trained models computational resources required should remain relatively low. However, if this is not the case, access to the Universities computer resources is available.

3.4.3 Skill Set

Currently I am familiar with deep learning methods through the research I have performed and how this can be integrated into my project effectively. I have recently refined my skill in PyTorch (Paszke et al., 2019) meaning implementing my solution should be feasible.

3.4.4 Conclusion

After considering the many aspects of my project, I have determined that the project is feasible to complete within the given time restrictions with my current skill level and understanding of the topic.

3.5 Out-of-Scope

- Real time processing: Given the limitations on time and computational resources, achieving a system that runs in real time is not a required goal.
- Training: Given the limitation on computational resources training will be kept to a minimum through using pre-trained modules. This will keep training requirements to a minimum and training a fresh model is not a goal of this project.
- Custom Model: As mentioned above, pre-trained models will be used to keep within time and computational limitations. Creating a custom model would be a research effort and is not the goal of this dissertation.
- Deployment: This project aims to produce richer more contextual dense captions. It is not a goal of this project to create a distributive product but to implement a potential improvement to current dense caption generation.

3.6 Evaluation

3.6.1 Datasets

Dense Captioning

For testing caption accuracy and relevance I will use the YouCook2 dataset as they are the standard in dense captioning research making my results comparable to previous solutions. The datasets provide annotations for both events and captions and preprocessed visual feature are readily available, decreasing finetuning and evaluation times. It should be noted that the set of YouCook2 videos used to evaluate the model will not be any of the ones used to finetune it.

Multi-Modal

For finetuning the decoder i will use the YouCook2 dataset. The choice to use this dataset is that it is freely available, a reasonable size and comes with audio that can be extracted from each video.

3.6.2 Metrics

- BLEU-1, BLUE-2, BLUE-3 and BLEU-4: Used to measure the matches of n-grams between the generated captions and the reference captions. This provides a precision oriented assessment.
- METEOR: Evaluates translation hypothesis by considering synonyms, stemming and exact matches. This helps in conveying the fluency and intent of the captions.

- CIDEr: Has a focus on semantic accuracy pf the generated captions by prioritizing semantically important words. This offers insights into the contextual alignment of captions with video content.
- Rouge-L: Based on the longest common subsequence, useful for measuring overlap in the sequence.
- Recall@x: Fraction of ground-truth segments correctly matched at IoU threshold x and time x.
- Precision@x: Fraction of predicted segments that are correct at IoU threshold x and time x.
- F1@x: Harmonic mean of precision and recall at IoU threshold x and time x.

Chapter 4

Design

4.1 Introduction to Design

In this chapter the design, structure and details of my project will be discussed. My implementation will build upon a previously implemented version of the Vid2Seq model Yang et al. (2023) in the VidChapters-7M project Miech et al. (2022). The implementation I use will be pre-trained on their VidChapters-7M dataset and fine-tuned with other datasets, providing me with some starting checkpoints. This will provide a solid, pre-trained base model for me to add an audio modality to and compare my results against.

4.2 System Architecture

4.2.1 VidChapters-7M Design

The Vid2Seq pipeline implemented by Miech et al. (2022) is composed of two encoders, ViT-L/14 by Radford et al. (2021) and Whisper-Large-V2 by Radford et al. (2023). These encoders produce visual features and ASR captions to be tokenised. These tokens are then passed into the T5 Model by Raffel et al. (2020) to produce dense captions as shown in Figure 4.1.

Whisper-Large-V2

The Whisper-Large-V2 model by Radford et al. (2023) is used to produce ASR captions from a 16kHz .wav file input. The speech is then processed, and output in the format:

$$S_i = \left(t_i^{\text{start}},\ t_i^{\text{end}},\ \text{text}_i\right) \tag{4.1}$$

The Whisper model was pre-trained by OpenAI and is used as a frozen encoder within the VidChapters-7M Vid2Seq implementation.

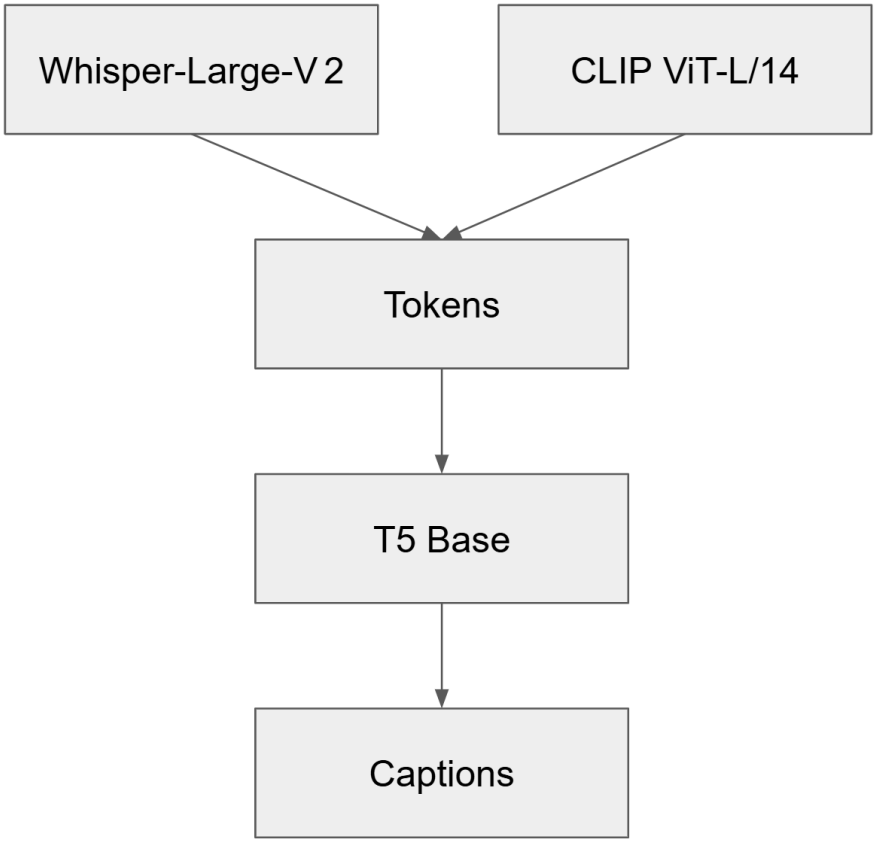

Figure 4.1: Module representation of VidChapters-7M Vid2Seq model.

CLIP ViT-L/14

The CLIP ViT-L/14 model by Radford et al. (2021) is used to extract visual features from sampled video frames. The version used is a pre-trained vision transformer (ViT) with a contrastive learning objective trained on large-scale image-text pairs, allowing the model to produce meaningful image feature sets. The implementation used in the VidChapters-7M Vid2Seq uniformly samples 100 frames from the video and passes them through the CLIP image encoder. The features are now produced as vectors [1, 100, 768], suitable to be turned into tokens.

Tokenization

Before the encoder outputs can be input into the T5 Model by Raffel et al. (2020), each modality must be transformed into a consistent token format. The text ASR output is firstly cleaned before being tokenised by the T5 tokenizer. This tokenizer uses SentencePiece-based subword tokenization to produce discrete token IDs that are compatible with the T5 models

vocabulary. However, due to the T5 models limited token input of 512, the ASR is only tokenized up to 411 tokens before being cut off. The 412th token in the sequence will then be the end-of-sequence token. For the visual features, there are minimal changes required as the CLIP ViT-L/14 model outputs the features in the correct dimensions. These visual token are then concatenated as pseudo tokens to make a token input length of 512. The unified token sequence now allows the T5 model to attend across the two modalities in a shared representation space.

T5 Model

The T5 model by Raffel et al. (2020) serves as the models decoder that generates event-level captions from a unified multimodal token input. After the tokens are input the decoder autoregressively generates event titles, treating the dense captioning task as a conditional text generation problem. The T5 model is pretrained on the Colossal Clean Crawled Corpus (C4) Raffel et al. (2020) and then further finetuned by the VidChapters team on datasets such as HowTo100M Miech et al. (2019), allowing the model to establish relationships between images and text.

4.2.2 Vid2Seq with Audio

For my implementation of including an audio modality, I will expand on the Vid2Seq architecture discussed above, implemented by Miech et al. (2022). The design of my framework will differ by having Wav2Vec 2.0 Base by Baevski et al. (2020) as an audio encoder alongside the encoders already used, and the token input ratio into the T5 model will also change, as shown in Figure 4.2.

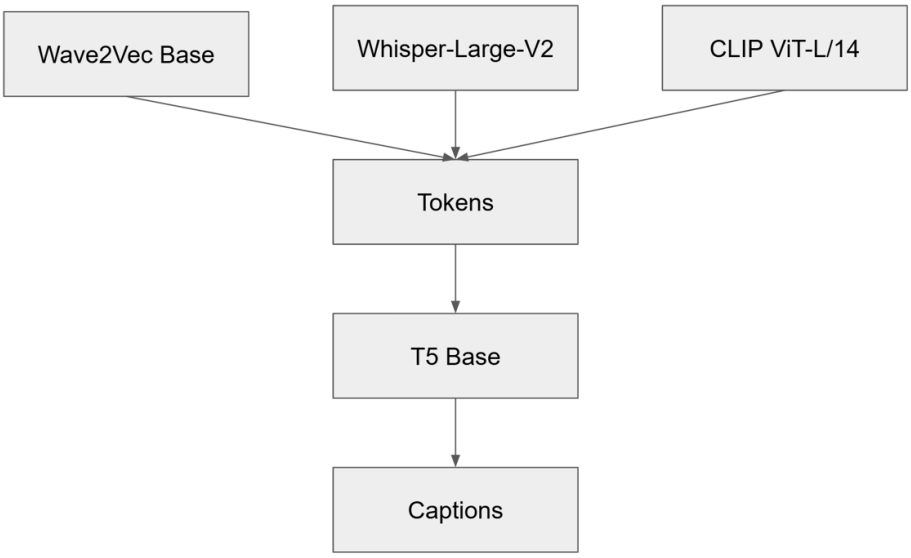

Figure 4.2: Module representation of audio Vid2Seq model.

Wav2Vec 2.0 Base

The Wav2Vec 2.0 Base model by Baevski et al. (2020) will be used to extract audio features from a .wav file input. The model is a self-supervised speech model trained on unlabeled audio data, designed to extract meaningful audio features. These features include pitch, tone, and rhythm which can be difficult to depict with visual features and text.The audio features are extracted from the .wav file in 20ms sections, with vector shape [1, 768].

Tokenization with Audio

The processing of ASR and visual tokens remains the same except for the ratio of tokens for each modality as shown in Figure 4.3. Pre-processing of the audio features is required to fit the 100 token limit. The approach chosen is to split the audio file into 100 chunks and average the audio features across this chunk producing 1 token. This allows for the audio features to align with the 100 visual feature vectors, adding more context. The audio tokens are then concatenated to the end of the token input already contain the ASR, end-of-sequence, and visual tokens.

Finetuning

After adding the audio features onto the T5 input and reducing the input of the ASR tokens, the T5 model will have to learn to interpret the new modality. Due to computational and time constraints the T5 model will be finetuned on the YouCook2 Zhou et al. (2018) dataset.

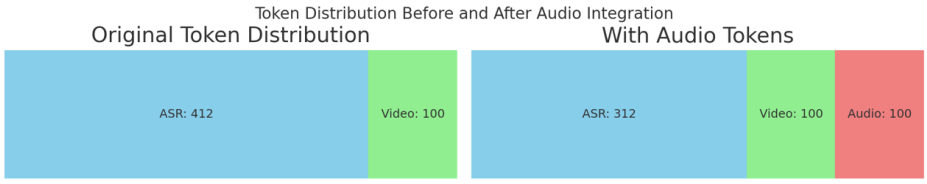

Figure 4.3: Token Ratios.

4.2.3 Input and Output Formats

The system processes video segments based on timestamped annotations and caption data provided by the dataset selected when the program is run. The datasets contain manually defined temporal segments paired with their corresponding caption. For each video segment, 100 uniformly extracted frames are used as the visual content. The visual features are precomputed using the CLIP ViT-L/14 model Radford et al. (2021) and are loaded as .npy files during evaluation.

The ASR caption transcripts are also pre-generated and provided as pkl files. These transcripts are loaded a tokenized using the T5 tokenizer Raffel et al. (2020) before being used as an input to the system.

In contrast, audio feature are extracted using a custom-made pre-processing script developed as a part of this dissertation. The script extracts 16kHz audio from the original video files and processes the audio with the Wav2Vec 2.0 Base model Baevski et al. (2020). The resulting features are then downsampled into 100 vectors using temporal pooling and are saved as .pt files to be loaded during evaluation or fine-tuning. The final model input consists of a sequence of 512 tokens composed of:

- 311 ASR tokens,
- An end-of-sequence token,
- 100 Visual pseudo-tokens from CLIP,
- 100 Audio tokens from Wav2Vec Base.

It should be noted that the ASR tokens include timestamps from when the speech is said in the .wav input as shown in Equation 4.2. The tokens are then passed as a single input to the T5 model, which autoregressively generates captions for each event. The final output is then a structured JSON file containing captioned events in the format:

$$\text{Segment}_i = \Big(\text{“sentence”}_i,\ \text{“timestamp”}_i = \Big[t_i^{\text{start}},\ t_i^{\text{end}}\Big]\Big) \tag{4.2}$$

The outputs are then evaluated using standard dense video captioning metrics, to assess caption quality and temporal accuracy.

4.2.4 Design Justification

Why Build on Vid2Seq?

The decision to use Vid2Seq Yang et al. (2023) as the base model for implementing my solution was largely due to its modularity and state of the art performance. The model is comprised of many sub-modules such as encoders, a tokenizer and a decoder stitched together to complete a dense captioning framework. This structure allows me to freely add encoder models easily and fine-tune existing models within the framework without any drastic changes or retraining being required. Another big factor contributing to its use is that it is documented very well and is available open source from Git Miech et al. (2022) with accompanying preprocessed data, making the limited computational power I have access to less of an issue.

Why Add Audio Modalities?

Adding a model such as Wav2Vec 2.0 Base by Baevski et al. (2020) to the base Vid2Seq pipeline, allowing audio features to impact the caption generation. The addition of audio features will allow the model to understand the context of events that were previously ambiguous through only text and visuals. While impacting caption quality, visual features can also help with event localization as prolonged sounds or shifts in background noise could mark different events happening.

Why Use These Encoders?

CLIP Vit-L/14 Radford et al. (2021) was implemented within the base Vid2Seq model as the visual encoder. I decided to keep the visual encoder the same, due to its compatibility with the already implemented pipeline. The encoder outputs its feature vectors in the format needed to be formatted directly into tokens, and only does it for the 100 frames needed. In terms of comparability, keeping this model also outlines the impact of audio features in the results instead of another visual encoder performing better or worse. Pre-processed visual data is also provided for each YouCook2 Zhou et al. (2018) video using this model, meaning I can save computational costs for pre-processing later.

Whisper-Large-V2 Radford et al. (2023) was also implemented in the base Vid2Seq Miech et al. (2022) model as the speech encoder. The model is pretrained on large real-world speech data, providing reliable and timestamped ASR transcriptions. The decision to keep using this model is similarly to keep the performance of the results based on the new audio features,and the fact that ASR transcriptions are also available pre-generated for the YouCook2 dataset.

Wav2Vec 2.0 Base Baevski et al. (2020) is implemented by me in the design as the audio encoder. The model was selected because of its balance between performance and size. It is trained on unlabeled audio data, making it suitable for extracting features from general audio. The model also consistently outputs its feature vectors in the correct token shape for tokenization, so only minimal pre-processing is required before tokenization.

Why Late Fusion?

The implementation of late fusion is due to the time constraints of the project. Late fusion allows for the model to interpret audio features along with ASR and visual features without major pre-processing before hand, while still maintaining some simplicity to implement. Although other fusion strategies, such as attention layers before tokenization may establish a better relationship between the modalities, it will take time to train these layers. The use of later fusion also remains consistent with the implementation of visual feature within VidChapters-7M version of Vid2Seq.

4.2.5 Challenges

Token Limits

The T5 model Raffel et al. (2020) has a strict token input limit of 512 tokens. The original Vid2Seq implementation Miech et al. (2022) made use of these tokens completely, meaning some redistribution was required to implement my 100 audio tokens. The new chosen token ratio implements 311 ASR tokens, 1 end-of-sequence token, 100 visual tokens ,and 100 audio tokens. This approach allows for the visual and audio tokens to align temporally, improving their contextual impact during caption generation. The decision to reduce the ASR token input amount was due to their implementation in the base Vid2Seq model, which implemented a hard cut-off at 311 tokens. The reduction of ASR tokens means that the videos captioned

will have to be shorter in length; otherwise, the ASR tokens will not be able to cover the entire video. The choice of token distribution will effect the models effectiveness for longer videos, but could theoretically be compensated using video chunking, carefully segmenting to avoid splitting coherent events..

Compute and Time Constraints

Due to the Vid2Seq pipeline containing many models, compute power and time constraints become a large concern. The access to pre-processed data provided by the previous implementation reduces these concerns; however, fine-tuning and audio extraction are two large tasks that have to be run locally. The restrictions this places on my implementation is that I will have to use the Wav2Vec 2.0 Base model Baevski et al. (2020), potentially missing out on deeper audio relationships. This issue also restricts the amount of finetuning I can perform on the model and entirely eliminates the option of retraining the whole pipeline. No retraining means that the relationship between ASR and visual features will be much deeper than the relationship with the audio features, something that finetuning a model trained on terabytes of ASR and visual features cannot help by being fine-tuned on 10's of gigabytes of audio data.

Fusion Simplicity

The decision of fusion method heavily relies of computational power available and the complexity required to implement it. The primary focus of this project was the implementation of an audio modality, a simple fusion strategy was implemented that was already compatible with the previous Vid2Seq implementation Miech et al. (2022). The challenge in implementing a complex fusion strategy is my limited PyTorch Paszke et al. (2019) experience and the training time required for such a strategy. The implementation of a simple solution reduces the capacity to model deeper audio-visual relationships or optimize encoders jointly.

4.2.6 Summary

The design presented in this chapter builds on the VidChapters-7M Miech et al. (2022) implementation of Vid2Seq Yang et al. (2023), extending it with a new audio modality using the Wav2Vec Base model Baevski et al. (2020). This addition was motivated by the limitations of relying solely on visual and ASR features, especially in scenarios where context, emotion, or ambient sound carry semantic weight. The modular architecture of Vid2Seq made it possible to integrate audio features with minimal disruption, and token-based late fusion ensured compatibility with the T5 Raffel et al. (2020) input constraints. Design choices were guided by both the aims of this dissertation and practical limitations around compute and training time. While some compromises were necessary, this design demonstrates the feasibility of integrating raw audio representations into a dense captioning pipeline and sets the foundation for further improvements in multimodal video understanding.

Chapter 5

Implementation and Testing

5.1 Environment and Tools

In my dissertation project I implemented two different Vid2Seq pipelines (Yang et al., 2023). One was for a general demo run, where the only thing you have to input is a .mp4 video file and dense captions are produced. This pipline uses the whisperx environment to run the demo_asr.py file which produces the ASR transcript, the Vid2Seq environment was then used to run the audio extraction script demo_audio_features.py file which extracts and saves the audio features, finally the Vid2Seq environment is then used to run the demo_vid2seq.py file which produces the dense captions.

The other was a fine-tuning and evaluation pipeline which made use of pre-processed data so elements like the speech and visual encoders don't need to run. The Vid2Seq environment was used to run all files related to this pipeline, including extract_audio_features_gpu.py and then dvc_audio.py.

5.1.1 Training and Evaluation Environment

The implementation was developed using Python 3.9.21 in a Conda-managed (Anaconda, Inc., 2023) environment named `vid2SeqTorch`. Core deep learning components included PyTorch 2.5.1 (Paszke et al., 2019) with CUDA 12.1 support, TorchAudio 2.5.1, and HuggingFace Transformers 4.28.0 for model loading and tokenization. The Wav2Vec 2.0 Base and Whisper-Large-V2 models were accessed through the Transformers library, while audio extraction from video files was handled using `ffmpeg` and the `ffmpeg-python` wrapper. Additional tools such as NumPy, pandas, and Matplotlib were used for numerical processing and visualisation. Dataset files, pretrained checkpoints, and extracted features were manually organised and referenced using configured paths within the pipeline. A full list of environment dependencies can be found in the project's `environment.yml` file.

Tool/Library	Version
Python	3.9.21
PyTorch	2.5.1 + CUDA 12.1
TorchAudio	2.5.1
Transformers (HuggingFace)	4.28.0
ffmpeg-python	0.2.0
NumPy	1.21.6
Pandas	1.3.4
Matplotlib	3.4.3
sentencepiece	0.2.0
OpenCV	4.5.5
scikit-learn	0.24.2

Table 5.1: Key tools and libraries used in the Vid2Seq implementation environment.

5.1.2 Demo Environment

In addition to the primary conda environment used for model development and evaluation, an environment called whisperx was used to extract ASR transcripts (Radford et al., 2023) from videos that don't already have this done. This environment was used in an end-to-end demo set-up used to demonstrate the framework working. The environment was built using Pyhton 3.10.16 and included the `transformers` (v4.48.3), `torchaudio`, `faster-whisper`, and `whisperx` libraries to support both standard and accelerated inference.

Tool/Library	Version
Python	3.10.16
PyTorch	2.5.1+cu121
Torchaudio	2.5.1+cu121
Transformers	4.48.3
WhisperX	3.3.1
Faster-Whisper	1.1.0
SentencePiece	0.2.0
FFmpeg (via av/ffmpeg)	14.1.0
Huggingface Hub	0.28.1
Tqdm	4.67.1
Pandas	2.2.3
NumPy	1.26.4
Matplotlib	3.10.0

Table 5.2: Key dependencies used in the WhisperX environment.

The WhisperX environment described above is specific to the demo pipeline; however, to run the main caption generation file, the conda environment (Anaconda, Inc., 2023) described in the training and evaluation section is required.

5.1.3 Hardware

The code was run on varying machines and software due to run-time and computational requirements. Firstly, the demo code for producing dense captions from a video used an RTX 3050 Laptop GPU with 4GB of VRAM. It should be noted that GPU memory required increases as video length increase, so the video chosen could impact run-time drastically. The demo code was run with this GPU on the OS Windows 11. For the training and evaluation of the model, an RX 7800XT with 12GB of VRAM was used on Linux Ubuntu 24.04. The GPU was interfaced using a docker container with PyTorch configured for ROCm to enable GPU acceleration. Despite the setup it should be noted that this approach was used due to resource availability and the code should run fine on Nvidea GPUS's in the conda environments described previously.

5.1.4 Reproducing My Results

When reproducing the contents of my project it should be noted that there are two versions available. To reproduce my models performance metrics follow `Fine-Tuning and Evaluation` and if you want to run a solution converting a given video to dense captions, follow `Demo`.

Fine-Tuning and Evaluation

For this implementation, only one environment was used and can be found detailed above in Table 5.1. The conda yaml file can also be found for the environment used in the code submitted under the name `vid2SeqEnv.yml`.

Once the environment is ready, this command can be run in the directory Fine Tuning and Evaluation or the Dir containing your code:

```
python dvc_audio.py \
  --presave_dir ./experiments \
  --save_dir youcook_audio_run \
  --device cuda \
  --model_name t5-base \
  --youcook_train_json_path ./YouCook2/train.json \
  --youcook_val_json_path ./YouCook2/val.json \
  --youcook_features_path ./YouCook2/clipvitl14.pth \
  --audio_features_train_path ./YouCook2/audio_features_train_768 \
  --audio_features_val_path ./YouCook2/audio_features_val_768 \
  --youcook_subtitles_path ./YouCook2/youcook2_asr_align_proc.pkl \
  --batch_size 4 \
  --batch_size_val 2 \
  --epochs 50 \
  --use_audio_features \
  --num_workers 4 \
```

```
  --combine_datasets youcook \
  --load ./vid2seq/vid2seq_htmchaptersyoucook.pth
```

This will make use of the pre-processed audio features I have provided for YouCook2 and will fine-tune using the `--load` checkpoints as its starting point. Once running the code will save the checkpoints at the end of each epoch to the `presave_dir` and display the evaluation results for that epoch. If you would like to run the whole solution from scratch then there are python scripts in the `YouCook2` folder to download and extract your own audio features.

To extract your own audio features, you will need to download the YouCook2 videos using `download_cook2_dataset_videos.py`. It is currently configure to just download the val videos, so once these are downloaded you will have to change the variables `DATASET_JSON_PATH` and `SAVE_VIDEOS_DIR` to the values shown here:

```
DATASET_JSON_PATH = "train.json"  # Path to your YouCook2 train.json
SAVE_VIDEOS_DIR = "youcook2_train_videos" # Save the downloaded videos
```

Listing 5.1: Manually truncating ASR tokens to fit with audio

Once all the videos are downloaded, the audio features can be extracted using `extract_audio_features_gpu.py`. Again this is only configured to extract for the training videos at the moment, so once it is done, you will need to change `video_folder` and `output_folder` to the values shown here:

```
video_folder = "YouCook2/youcook2_val_videos/"
output_folder = "YouCook2/audio_features_val_768/"
```

Listing 5.2: Manually truncating ASR tokens to fit with audio

If you would like to skip the fine-tuning stage and evaluate using checkpoints I have provided, you can use the command:

```
python dvc_audio.py \
  --presave_dir ./experiments \
  --save_dir youcook_audio_eval \
  --device cuda \
  --model_name t5-base \
  --youcook_train_json_path ./YouCook2/train.json \
  --youcook_val_json_path ./YouCook2/val.json \
  --youcook_features_path ./YouCook2/clipvitl14.pth \
  --audio_features_train_path ./YouCook2/audio_features_train_768 \
  --audio_features_val_path ./YouCook2/audio_features_val_768 \
  --youcook_subtitles_path ./YouCook2/youcook2_asr_align_proc.pkl \
  --batch_size 2 \
  --batch_size_val 2 \
  --epochs 1 \
  --use_audio_features \
```

```
  --num_workers 4 \
  --combine_datasets youcook \
  --load ./experiments/youcook_audio_run/continued_checkpoint_epoch_0.pth \
  --eval
```

This invokes the `--eval` argument and evaluates the `--load` checkpoints on the val data split. The result for each metric will be output together in the terminal.

Demo

The demo was implemented as an end-to-end example of the model being implemented. To run the demo two environments are required, the Whisperx environment shown above and the Vid2Seq environment shown above. Both have a conda environmnet yaml configuration file in the code folders named `vid2SeqEnv.yml` and `whisperx.yml`.

Firstly, you need to download a video to the working directory and set you environment variable using the command `export TRANSFORMERS_CACHE=./TRANSFORMER_CACHE`. Then in the whisperx environment run:

```
python demo_asr.py \
  --video_example sample_video.mp4 \ # Use name of your video
  --asr_example aligned_asr.pkl \ # Name of ASR file output
  --device cuda
```

Now you have you ASR transcript. Switch to the vid2Seq environment and run `demo_audio_features.py` where `sample_video.mp4` is the name of your video file:

```
python demo_audio_features.py sample_video.mp4
```

Then while in the same environment, run:

```
python demo_vid2seq.py \
  --video_example sample_video.mp4 \ # Your video here
  --asr_example aligned_asr.pkl \  # Your asr here
  --load ./vid2seq_audio_finetuned.pth \ # Your checkpoints here
  --device cuda \
  --model_name t5-base \
  --max_feats 32 \
  --num_bins 100 \
  --num_beams 4 \
  --max_input_tokens 512 \
  --max_output_tokens 512 \
  --top_p 1.0 \
  --repetition_penalty 1.2 \
  --length_penalty 0.8 \
  --seed 42
```

The audio features will then be loaded into the script automatically from `audio_features.pt`. The captions will be output to the terminal.

5.2 Audio Pre-processing

5.2.1 Demo Audio Pre-processing

The demo pipeline requires audio pre-processing to be carried out on a single video, extracting audio features in a format suitable for integration with the Vid2Seq model (Yang et al., 2023). This task is handled by the demo_audio_features.py script, which performs 3 main functions: (1) audio extraction from an .mp4 video, (2) feature extraction using a pre-trained Wav2Vec 2.0 Base model (Baevski et al., 2020), and (3) downsampling the resulting feature matrix to a suitable size for input to the model.

Audio was extracted from the mp4 videos using ffmpeg, where each video was converted into a mono-channel waveform sampled at 16kHz. This is required by the Wav2Vec 2.0 Base model to perform a valid feature extraction. The following code performs the audio extraction:

```
ffmpeg.input(video_path).output(wav_out_path, ac=1, ar=16000).run
(overwrite_output=True)
```

Listing 5.3: Extracting mono 16kHz audio using `ffmpeg`

Once audio was extracted, the script loaded the pretrained Wav2Vec base model and its corresponding feature extractor from HuggingFace Transformers. These were responsible for processing raw waveforms into token-level embeddings.

After the audio has been extracted. the script loads the pretrained Wav2Vec 2.0 Base model and its corresponding feature extractor from HuggingFace Transformers (Wolf et al., 2020). These are responsible for processing raw wav files into token-level embeddings.

```
processor = Wav2Vec2FeatureExtractor.from_pretrained(
    "facebook/wav2vec2-large")
model = Wav2Vec2Model.from_pretrained("facebook/wav2vec2-large")
model.eval()
```

Listing 5.4: Loading the Wav2Vec2 model and processor

To handle long inputs, the waveform is split into non-overlapping 10-second segments. If a segment is shorter than 10 seconds (the final segment), it is padded with zeros to maintain a consistent shape.

```
chunk = waveform[:, i:i+chunk_size]
if chunk.shape[-1] < chunk_size:
    chunk = F.pad(chunk, (0, chunk_size - chunk.shape[-1]))
```

Listing 5.5: Handling chunking and padding of waveform

The process of feature extraction is then performed on each segment. The resulting hidden states form the Wav2Vec Base model (Baevski et al., 2020) are trimmed to retain only the first 768 dimensions (already satisfied by the base model). This maintains consistency with the visual features used as tokens.

```
output = model(input_values).last_hidden_state.squeeze(0)[:, :768]
```

Listing 5.6: Extracting 768-dim features from Wav2Vec output

Finally, the output of feature matrix is downsampled to 100 tokens using mean pooling across uniform temporal segments. This ensures token constraints are not exceeded and varying video length can be handled. Now a tensor of shape [100, 768] is produced, which is saved in a .pt format using PyTorch (Paszke et al., 2019) for later use in the Vid2Seq model (Miech et al., 2022).

```
pooled.append(segment.mean(dim=0))
return torch.stack(pooled)  # shape: (100, 768)
```

Listing 5.7: Downsampling audio features to 100 tokens

```
torch.save(features, "audio_features.pt")
```

Listing 5.8: Saving extracted features to a PyTorch file

This pre-processing step allowed audio features to be seamlessly integrated as a third modality alongside vision and ASR transcripts in the Vid2Seq pipeline (Yang et al., 2023).

5.2.2 Fine-Tuning and Evaluation Audio Pre-processing

To prepare the audio input for the fine-tuning and evaluation of my multimodal Vid2Seq, two scripts were used. The first script download_cook2_dataset_videos.py, downloads the YouCook2 video files base on youtube IDs provided in the datasets JSON files. This was required because audio isn't used in dense captioning so the YouCook2 dataset doesn't come with seperate audio files. The second, extract_audio_features_gpu.py, extracts and processes the audio into fixed-length token sequences to be used in the model.

To download raw video data, the script uses yt-dlp to fetch and store youtube videos in a .mp4 format. The script attempts to download all videos but if it is not available, has been privatized, or exists locally, it is skipped.

```
subprocess.run([
    YTDLP_PATH,
    "-f", "best[ext=mp4]/best",
    "--output", save_path,
    youtube_url
], check=True)
```

Listing 5.9: Downloading YouCook2 videos using video IDs

Once the videos have been downloaded and can be accessed locally, each videos's audio is extracted and resampled to mono-channel 16kHz using ffmpeg. This is the format expected as input by the Wav2Vec 2.0 Base model (Baevski et al., 2020).

```
ffmpeg.input(video_path).output(wav_path, ac=1, ar=16000)
    .overwrite_output().run(quiet=True)
```

Listing 5.10: Extracting 16kHz mono audio from a video

The audio extracted from the videos is then passed through the Wav2Vec 2.0 Base model to generate a sequence of feature vectors. Each feature vector represents 20ms of audio in 768-dimensional space.

```
inputs = processor(waveform.squeeze(), sampling_rate=16000,
    return_tensors="pt")
input_values = inputs.input_values.to(device)
outputs = model(input_values).last_hidden_state.squeeze(0)
```

Listing 5.11: Extracting Wav2Vec2 audio features

The output features are then downsampled into 100 audio tokens by splitting the sequence uniformly and applying mean-pooling. The shape [100, 768] is crucial, as it is the required shape for the tokens to be input into the model. The final tensor is then saved in a .pt format using PyTorch (Paszke et al., 2019).

```
segment = features[start:end]
pooled_features.append(segment.mean(dim=0))  # Average pooling
return torch.stack(pooled_features)  # (100, 768)
```

Listing 5.12: Downsampling audio to 100 tokens

```
torch.save(features_100x768, output_path_trimmed)
```

Listing 5.13: Saving processed audio features

The audio files are all save with their ID as their name in the directory YouCook2/audio_features_val_768, and later accessed by the model during fine-tuning or evaluation.

5.3 Model Integration

This project is built on the Vid2Seq architecture (Yang et al., 2023), a multimodal transformer-based dense caption generation model using visual and speech features. The model has been extended in this work (Miech et al., 2022) to also incorporate audio features extracted using the Wav2Vec 2.0 Base model (Baevski et al., 2020).

Vid2Seq is built using multiple models, these include, a T5 encoder-decoder (Raffel et al., 2020) for text generation, a Vit-L/14 CLIP-based visual encoder (Radford et al., 2021), and a Whisper-Large-V2 speech encoder (Radford et al., 2023). These features are embedded as tokens, concatenated, and input in the T5 decoder for caption generation.

To enable the model to support audio as a third modality, the Vid2Seq class in vid2seq.py was modified. The change allows the model to take a third input stream: a [100,768] tensor representing down-sampled audio features from the Wav2Vec 2.0 Base model.

```
self.use_audio = True
```

Listing 5.14: Enabling audio input in the model

The forward() and generate() methods were extended to accept the new audio input tensor and a corresponding attention mask:

```
if self.use_audio and audio is not None:
atts_audio = torch.ones(audio.size()[:-1], dtype=torch.long).to(
    audio.device)
hidden_states.append(audio)
attention_masks.append(atts_audio)
```

Listing 5.15: Adding audio embeddings in the forward pass

The attention masks are concatenated alongside visual and speech tokens:

```
final_hidden_state = torch.cat(hidden_states, dim=1)
encoder_atts = torch.cat(attention_masks, dim=1)
```

Listing 5.16: Multimodal fusion via concatenation

To ensure that the tokens input to the T5 model are compatible, there has to be a strict limit of 512 total tokens per input. This is satisfied by having the ASR token amount calculated dynamically and reserving 100 tokens for audio. The visual tokens are already considered in the args.max_input_tokens, therefore reserving 311 tokens for ASR, 1 for the end-of-sequence token, 100 for visual tokens, and 100 for audio tokens.

```
TEXT_TOKEN_LIMIT = args.max_input_tokens - 100
input_tokens = input_tokens[:TEXT_TOKEN_LIMIT - 1]
input_tokens = torch.cat([input_tokens, torch.LongTensor(
    [tokenizer.eos_token_id])], 0)
```

Listing 5.17: Token truncation before fusion

For training, no updates were made to the CLIP or Whisper encoders, only the Vid2Seq model was fine-tuned. This was guaranteed by providing pre-extracted visual, audio, and ASR features as inputs. The fine-tuning was conducted using the dvc_audio.py script, which relies on dvc_dataset_audio.py when loading all three modalities.

Fine-tuning was performed using the default loss function of the T5 model. T5 implemented a cross entropy on the predicted tokens of the decoder, with respect to the ground-truth captions. This directly optimizes caption quality without introducing auxiliary objectives such as proposal generation or temporal alignment losses.

Training input handling is consistent with:

```
if args.use_audio_features and "audio" in batch_dict:
audio_features = batch_dict["audio"].to(device)
...
output = model(video=video, input_tokenized=input_tokenized,
output_tokenized=output_tokenized, audio=audio_features)
```

Listing 5.18: Dataloader preprocessing for audio

The tokenizer used was based on T5 and extended by the VidChapter-7M team with additional time tokens for temporal alignment:

```
new_tokens = ["<time=" + str(i) + ">" for i in range(num_bins)]
tokenizer.add_tokens(list(new_tokens))
self.t5_model.resize_token_embeddings(len(tokenizer))
```

Listing 5.19: Extending the tokenizer with time tokens

These temporal tokens allow the model to understand the alignment between audio, speech, and visual features across the video timeline.

The changes made to the model logic here applies to both the demo and fine-tuning pipelines, as the input handling and fusion strategy are identical. The only difference is that for the fine-tuning and evaluation pipeline features are pre-extracted, whereas in the demo pipeline the encoders are all used to extract features.

5.4 Code Challenges

Integrating a third modality into the Vid2Seq architecture introduced a number of implementation challenges. These included managing token constraints, ensuring temporal consistency across modalities, and extending input handling across the original model and data loader.

5.4.1 Token Truncation Logic

The original Vid2Seq (Yang et al., 2023) model was designed with two modalities in mind (vision and ASR), with total tokens allowed to be input capped at 512. Therefore, adding audio required the reservation of 100 tokens for Wav2Vec 2.0 Base (Baevski et al., 2020) features, which meant a dynamic truncation of ASR tokens needed implementing. Debugging this implementation was difficult due to silent crashes and errors caused by exceeding the token limit.

```
TEXT_TOKEN_LIMIT = args.max_input_tokens - 100
input_tokens = input_tokens[:TEXT_TOKEN_LIMIT - 1]
input_tokens = torch.cat([input_tokens, torch.LongTensor(
    [tokenizer.eos_token_id])], 0)
```

Listing 5.20: Manually truncating ASR tokens to fit with audio

Without introducing a dynamic solution, longer ASR transcripts would cause issues by exceeding the allowed token limit. This would then ultimately cause errors at the finetuning and inference stages.

5.4.2 Audio-Video Feature Mismatch

Initially when implementing feature extraction from the audio file, I set a limit of 30 seconds for audio length to test whether the code would run as expected. After confirming the code ran correctly, I forget to remove the 30 second limit and used audio features only extracted from the start of the video when evaluating my model. This issue caused the audio and visual features to be temporally misaligned and had to be caught in code review due to no errors being thrown.

5.4.3 Feature Shape Errors During Concatenation

During the early stages of the dissertation, the incorrectly shaped audio tensors caused runtime errors when attempting to concatenate them with the other tokens. The model expects a [B,T,D] shape but was being passed a [T, D] shape. I also had to ensure the attention masks for the audio tokens matched in shape for the fusion step.

5.4.4 Memory Constraints

The memory consumption for this implementation increased compared to the base Vid2Seq (Miech et al., 2022) implementation due to the added audio features. When running scripts, the batch size had to be conservative due to the memory differences when longer videos appeared, leading to VRAM errors. This was supplemented by using smaller batch sizes when fine-tuning and evaluating the model, and by using a graphics card with more memory.

5.4.5 Limited Dataset

When downloading the videos for YouCook2 (Zhou et al., 2018) it became apparent that not all the videos originally used were available still. To prevent the downloading script from crashing, IDs without a video were skipped. When the IDs were met in the finetuning stage a 0 vector was implemented as the audio tokens. This was implemented due to the small amount of data already available and it mimics processing a video with no available audio.

5.5 Testing Strategy

5.5.1 Evaluation Scheme

As stated in chapter 3, the main goal of this project is to improve caption quality and detail by incorporating audio features into the dense captioning pipeline. This section highlights the evaluation strategies used to access the success of my implementation.

When comparing the caption and temporal quality, several metrics were considered. This project focuses primarily on:

- BLEU-1 to BLEU-4: Measures n-gram precision by computing overlapping word sequences between generated and reference captions.
- METEOR: Measures semantic similarity by considering synonyms, word stems, and word order. Good for tasks where variation in language used is expected.
- CIDEr: Measures the contents relevance by weighting n-gram using TF-IDF. This places more value on semantically important or distinctive words.
- ROUGE-L: Measures the longest common subsequence between the predicted and base truth captions. Outlining structure and fluency.
- Recall@x / Precision@x / F1@x (at IoU thresholds): Assesses the temporal alignment between the predicted event boundaries and the ground truth ones. This is measured at different IoU strictness levels. F1 measures a balance between recall and precision.
- Recall@xs / Precision@xs / F1@xs (for fixed segment durations): Assesses how well the model captures events of specific lengths. This represents the temporal coverage across the video well.

5.5.2 Experimental Data Setup

The dataset I decided to use in this project is YouCook2 (Zhou et al., 2018). The dataset is well established for dense video captioning benchmarking and each video can have their audio extracted. The dataset contains instructional cooking videos, each annotated with event boundaries and captions that describe these events throughout the video. The acess to this data makes YouCook2 ideal for training dense captioning solutions.

There were 1333 videos used as training data and 457 for validation. All evaluations were conducted on the same validation set, which shared no videos with the training set. This was done to prevent a biased assessment of the models performance.

The data was prepared for model input using pre-processing. Visual feature were extracted by the CLIP ViT-L/14 model (Radford et al., 2021) in advance and provided in a file. These video frames were uniformly selected and resized to consistent size before feature extraction. The audio files were extracted from the YouCook2 .mp4 files and used to generate audio feature vectors using Wav2Vec 2.0 Base (Baevski et al., 2020). These feature vectors were then mean pooled into 100 feature vectors and saved as .pt files, allowing for efficient loading. The speech transcriptions were pre-generated using Whisper, which provided provided the ASR tokens (Radford et al., 2023). All modalities are then temporarily aligned based on the video duration and discretized into fixed-length bins.

The videos each include multiple captioned events, each associated with a start and end time token. These temporal token pairs are calculated to indicate the position in the video.

These two time tokens contain the tokenized caption and are padded or truncated to satisfy the token limit.

Evaluation was performed using the custom dvc_audio.py script. This script is an expanded on version of one implemented in the base Vid2Seq model. The script handles validation inference and metric calculation using imported functions. Predictions and ground-truth annotations are loaded by the script, outputs as a standard format, and calculates performance metrics. The dataset loading and batching were handled through dvc_dataset_audio.py, which ensured that the audio features were correctly included in each batch.

5.5.3 Baseline vs Extended Model

The baseline Vid2Seq model (Miech et al., 2022) used in this project combines visual features and ASR text to generate dense captions. It uses frame level features extracted by ViT (Radford et al., 2021) and transcribed speech with time tokens using Whipser (Radford et al., 2023). These modalities are temporally aligned and input into the T5 decoder (Raffel et al., 2020), generating timestamped captions for each event in the video.

To evaluate my solution of implementing audio features, an extended version of the model was implemented. The extended model uses audio features extracted from the video using Wav2Vec 2.0 Base (Baevski et al., 2020), with existing visual and ASR inputs. The audio features help capture background sounds, music, ambient noise, and paralinguistic signals. These are often lost in ASR transcription, but are important for context, emotion, and event discrimination. The audio features used were downsampled to a fixed sequence of 100 feature vectors per video and were concatenated as an additional token set to the T5 decoder.

In order to ensure a non biased comparison between the baseline and extended model, all the non-audio modules of the system were kept constant. The models both used the same tokenizer, the same number of frames per video, the same ground-truth time stamps and captions, and identical evaluation parameters such as batch size. This controlled setup allows us to credit any key differences in performance to the implementation of an audio modality.

This comparative setup allow for a direct assessment of how much caption quality, and contextual gain their is to be provided by the implementation of audio features.

Chapter 6

Results and Discussion

The evaluation of the models was performed using the 457-video validation split of the YouCook2 dataset (Zhou et al., 2018). The metrics used assess caption quality (such as METEOR, CIDEr, BLEU, and ROUGE-L), and temporal alignment (such as Recall@k and F1@k at various IoU thresholds and segment durations). The results compare the model across multiple fine-tuning eopchs as well as against the baseline and are shown in Table 6.1.

Table 6.1: Comparison of model performance across fine-tuning epochs and baseline (Vision + ASR only)

Metric	Epoch 1	Epoch 10	Epoch 20	Epoch 30	Baseline
METEOR	0.1202	0.1182	0.1142	0.1151	0.1230
CIDEr	0.6498	0.6510	0.5845	0.5867	0.6719
ROUGE-L	0.1439	0.1449	0.1346	0.1377	0.1493
BLEU-1	0.1636	0.1670	0.1549	0.1537	0.1672
BLEU-2	0.1159	0.1153	0.1073	0.1063	0.1182
BLEU-3	0.0820	0.0798	0.0737	0.0740	0.0839
BLEU-4	0.0562	0.0550	0.0489	0.0497	0.0579
Recall@0.3	0.6951	0.6640	0.6390	0.6653	0.6874
F1@0.3	0.6478	0.6552	0.6257	0.6504	0.6545
Recall@0.5	0.4346	0.4056	0.3888	0.4033	0.4370
F1@0.5	0.4066	0.4033	0.3830	0.3924	0.4163
Recall@0.7	0.2056	0.1849	0.1705	0.1753	0.2039
F1@0.7	0.1968	0.1850	0.1691	0.1723	0.1983
Recall	0.3430	0.3219	0.3060	0.3184	0.3407
F1	0.3218	0.3193	0.3008	0.3109	0.3260
Recall@1s	0.1709	0.1552	0.1507	0.1578	0.1649
F1@1s	0.1571	0.1550	0.1473	0.1515	0.1564
Recall@3s	0.4428	0.4002	0.3993	0.4043	0.4310
F1@3s	0.4070	0.3959	0.3867	0.3900	0.4036
Recall@5s	0.6064	0.5517	0.5375	0.5505	0.5946
F1@5s	0.5559	0.5412	0.5245	0.5316	0.5541
Recall@10s	0.8082	0.7528	0.7516	0.7548	0.7896
F1@10s	0.7565	0.7359	0.7323	0.7358	0.7511
Recall@30s	0.9528	0.9574	0.9489	0.9500	0.9505
F1@30s	0.9433	0.9464	0.9427	0.9408	0.9426
Recall@60s	0.9851	0.9888	0.9848	0.9876	0.9814
F1@60s	0.9839	0.9859	0.9861	0.9864	0.9842

The baseline checkpoints used in this project were provided by the VidChapter-7M team (Miech et al., 2022). It was pre-trained large-scale datasets including HowTo100M (Miech et al., 2019), VidChapters, and YouCook2 (Zhou et al., 2018). The extended model used Wav2Vec 2.0 Base (Baevski et al., 2020) to extract audio features that were added to the input pipeline. Fine-tuning of this model was performed using the 1333-video training split from YouCook2, with metric results recorded at epochs 1, 10, 20 and 30.

Overall the inclusion of an audio modality led to improvement in several key temporal metrics. At Epoch, the Recall@0.3 score was 0.6951, slightly outperforming the baseline score of 0.6874. F1@5s also saw a consistent improvement in early epochs, indicating that event segmentation was more accurate when audio context was available. These results support the

idea that non-verbal audio features can enhance video understanding and temporal precision.

Semantic metrics for caption quality such as METEOR and CIDEr also showed early improvements. METEOR reached 0.1202 at Epoch 1, while CIDEr peaked at 0.6510 by Epoch 10, both approaching the baseline values. This pattern suggests that audio contributed to caption quality earlier in the training.

However, in later epochs semantic performance declined. For example, BLEU scores and ROUGE-L dropped after Epoch 10. This indicates possible overfitting or diminishing returns from further fine-tuning on a small dataset. This implies that while audio may be beneficial, it could require more careful integration to maintain a positive impact.

The extended model underperformed the baseline in areas related to caption fluency and lexical diversity. This became largely apparent in later epochs and could be due to token limit constraints imposed by the T5 model (Raffel et al., 2020). The imposed constraint required changing the input token ratio and reducing the ASR input. As a result useful information from a given modality could be missed.

The extended results show that integrating audio features improves temporal segmentation and supports better performance on some short segment tasks. These benefits shown are most prevalent in the earlier epochs, but as fine-tuning continues performance tapers off. This is likely due to model constraints and dataset limitations. These results enforce the value of multimodal inputs for dense captioning, while highlighting the importance of training strategies and architecture scaling for maintained improvements.

The outcomes of this project can be evaluated by revisiting the aims and objectives outline in Chapter 3. The main aim was to improve the quality and accuracy of dense video captions by incorporating audio features as a third modality.

In line with these aims, the implemented pipeline successfully built upon the existing Vid2Seq model (Yang et al., 2023). This was achieved using Wav2Vec 2.0 Base (Baevski et al., 2020) as an encoder to extract audio features, which once encoded were passed into the multimodal architecture. The model was implemented without requiring retraining from scratch, making efficient use of the available resources. The evaluation shows that the extended model produced improved scores in several temporal metrics when compared to the baseline. Indicating that the audio features contributed meaningfully to the temporal alignment in this implemented pipeline.

The objective of extracting audio feature such as speech and non-verbal audio, was met using Whisper (Radford et al., 2023) for ASR and Wav2Vec 2.0 Base (Baevski et al., 2020) for general audio. These features were aligned with visual features through temporal binning, meeting the second objective of synchronizing visual and audio features to enhance caption generation.

While my solution showed modest improvements in certain evaluation metrics, there are several directions future work could take to enhance the work that has been completed. One key limitation lies in the audio sampling strategy. The approach I implemented uses mean pooling to produce fix-length representations of audio features covering an entire video. Although this ensures global coverage, it may smooth out high-intensity audio cues, such as

clapping or variable sounds related to specific events. Future approaches could implement an extraction method that focuses around local peaks or fluctuations in the audio, potentially capturing more meaningful audio-visual correlations.

Another challenge introduced by adding audio as an additional modality was the constraint imposed by the 512 token limit. This restricts the amount of multimodal information that can be input into the T5 model (Raffel et al., 2020) per video. A potential solution to this issue would be to segment the video into shorter segments. These segments could then be processed individually, with context from the previous segment passed into the next. This would maintain narrative continuity without exceeding input length limits. This solution would also have to implement a pre-process that could segment the video without splitting events.

Resource limitations also affected the models scalability. The constraints imposed by limited memory and computational power meant larger models or retraining was not feasible. With access to more powerful hardware, future work could implement the pipeline with more powerful encoders, jointly train all components, or train using large batch sizes.

Finally, dataset size and diversity are areas for improvement. The fine-tuning implemented in this project was conducted using YouCook2 (Zhou et al., 2018) due to storage and time constraints. Using larger and more diverse datasets such as VidChapters (Miech et al., 2022) or ActivityNet Captions (Caba Heilbron et al., 2015) could improve the model's generalization and enhance its responses to variations in scenes.

Chapter 7

Conclusion

This dissertation aimed to investigate whether augmenting dense video captioning with video features could improve contextual accuracy and temporal alignment. The solution built upon the Vid2Seq model (Yang et al., 2023) by incorporating audio as a third modality extracted using the Wav2Vec 2.0 Base (Baevski et al., 2020) encoder. These features were processed and integrated alongside visual (Radford et al., 2021) and ASR (Radford et al., 2023) inputs, in order to improve caption generation. This was especially relevant when the audio information was of significant narrative weight.

The results demonstrated that while semantic caption quality showed slight degradation, the audio-augmented model showed consistent gains in temporal metrics at shorter time intervals.

The findings suggest that the integration of the audio modality was effective in specific contexts, supporting the hypothesis that multimodal fusion can strengthen temporal segmentation. However, the evaluation results also highlighted limitations, such as the token limit on the T5 model (Raffel et al., 2020). This forced trade-offs in how the modalities were balanced on input to the model. The truncated ASR inputs may have reduced the fluency and completeness of the generated captions.

Furthermore, model scalability was limited by computational resources and time constraints. Due to memory limitations, a full retraining of the model was not feasible and fine-tuning was performed instead. Despite this inconvenience, the model still showed improvements in temporal segmentation with shallow integration of audio features.

From a broader perspective, this implementation successfully met many of the projects objectives. It showed that:

- An existing dense captioning model can be extended to incorporate audio features.
- Temporal and contextual alignment can be improved without extensive retraining.
- Evaluation on a standard dataset (YouCook2) (Zhou et al., 2018) showed that the model remains competitive, even with limited compute and data.

Although the implemented model showed some success, there are still important avenues to be explored. A more advanced audio sampling strategy could replace mean pooling with an approach that selects around peaks and troughs in the waveform. Likewise, dynamic segmentation of the video into smaller, contextually coherent clips could allow better use of the T5 input tokens. Finally, using a larger and more diverse dataset, such as VidChapters (Miech et al., 2022), could enhance the models generalization to unique contexts.

In summary, this project provides a practical implementation and promising evidence that the audio modality can improve dense captioning performance. Although further improvements in model design, data coverage, and training strategy are needed, the results underline the value of audio as a resource in the task of dense captioning.

Bibliography

Anaconda, Inc. (2023). Anaconda software distribution. https://www.anaconda.com.

Baevski, A., Zhou, H., Mohamed, A., and Auli, M. (2020). Wav2vec 2.0: A framework for self-supervised learning of speech representations. *arXiv preprint arXiv:2006.11477*.

Caba Heilbron, F., Escorcia, V., Ghanem, B., and Niebles, J. C. (2015). Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 961–970. IEEE.

Krishna, R., Hata, K., Ren, F., Fei-Fei, L., and Niebles, J. C. (2017). Dense-captioning events in videos. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 706–715. IEEE.

Miech, A., Alayrac, J.-B., Sivakumar, A., Laptev, I., Sivic, J., and Zisserman, A. (2022). Vidchapters-7m: Labeled chapter structure in 7 million videos. *arXiv preprint arXiv:2206.00247*.

Miech, A., Zhukov, D., Alayrac, J.-B., Tapaswi, M., Laptev, I., and Sivic, J. (2019). Howto100m: Learning a text-video embedding by watching hundred million narrated video clips. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2630–2640. IEEE.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., and Desmaison, A. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pages 8024–8035.

Radford, A., Gao, J. W., Clark, J., Gokaslan, A., and Zoph, B. (2023). Robust speech recognition via large-scale weak supervision. *arXiv preprint arXiv:2212.04356*.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. (2021). Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Venugopalan, S., Rohrbach, M., Donahue, J., Mooney, R., Darrell, T., and Saenko, K. (2015). Sequence to sequence: Video to text. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 4534–4542. IEEE.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3156–3164. IEEE.

Wang, Y., Zhu, L., Yang, Y., and Wu, F. (2021). End-to-end dense video captioning with parallel decoding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Virtual Conference. IEEE.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., and Brew, J. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45.

Xu, J., Mei, T., Yao, T., and Rui, Y. (2019). Msr-vtt: A large video description dataset for bridging video and language. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5288–5296. IEEE.

Yang, A., Nagrani, A., Seo, P. H., Miech, A., Pont-Tuset, J., Laptev, I., Sivic, J., and Schmid, C. (2023). Vid2seq: Large-scale pretraining of a visual language model for dense video captioning. *arXiv preprint arXiv:2302.14115*.

Ye, H., Huang, D.-A., Lu, Y., Yu, Z., Ping, W., Tao, A., Kautz, J., Han, S., Xu, D., Molchanov, P., and Yin, H. (2023). X-vila: Cross-modality alignment for large language models. *arXiv preprint arXiv:2405.19335*.

Zhou, L., Xu, C., and Corso, J. J. (2018). Towards automatic learning of procedures from web instructional videos. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32. AAAI.