VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

FINAL PROJECT

COURSE NAME: STATISTICAL MACHINE LEARNING

Student:
Vu Minh Phat (21127739)
Trieu Gia Huy (22127166)
Vo Thanh Nghia (22127295)
Vo Thinh Vuong (22127464)

Lecturer:
Dr. Ngo Minh Nhut
TA: Le Long Quoc

August 24, 2025

Contents

1 Information

1.1 Student Information

Student ID	Full name	Email
22127166	Trieu Gia Huy	tghuy22@clc.fitus.edu.vn
22127295	Vo Thanh Nghia	vtnghia22@clc.fitus.edu.vn
22127464	Vo Thinh Vuong	vtvuong22@clc.fitus.edu.vn
21127739	Vu Minh Phat	vmphat21@clc.fitus.edu.vn

1.2 Source Code

GitHub

2 Introduction

2.1 Background and Motivation

In the current digital age, we are witnessing an explosion of multimedia data, especially videos. From social media platforms like YouTube, TikTok, and Facebook to security surveillance systems, online courses, and personal archives, video has become a popular and effective medium for conveying information. However, the vast amount of information contained within these videos presents a major challenge: how can we access and retrieve specific information quickly and accurately?

Traditional search methods typically rely on human-created **metadata**, such as titles, descriptions, or tags. This approach is limited when users want to find specific details inside a video's content, such as a particular action, an object that appears for only a few seconds, the color of an item, or a sentence spoken at a specific moment. Watching an entire video just to find one piece of information is extremely time-consuming and inefficient.

To solve this problem, Artificial Intelligence (AI) technologies, particularly in the fields of **Computer Vision (CV)** and **Natural Language Processing (NLP)**, have introduced groundbreaking solutions. Models that can "understand" the content of images and videos (Image/Video Captioning) and "listen" to audio (Speech Recognition) are becoming increasingly powerful and precise.

Driven by this practical need, this project proposes the development of an **intelligent chatbot system that allows users to directly ask questions about the content of a video**. Instead of watching the entire video, users can interact with the system using natural language, asking questions and receiving accurate answers based on both the visual and audio content of the video.

2.2 Project Objectives

This project aims to solve the problem of detailed information retrieval from videos by building and integrating an intelligent chatbot system. The specific objectives are as follows:

Build a Comprehensive Context for Videos: To automatically extract and combine information from four primary sources:

- **Static Visual Content (Image Captioning):** Summarize the general scene, main objects, and their attributes from key frames of the video.
- **Action-based Content (Video Captioning):** Describe the actions and events happening in the video using a specialized model (**BTKG** [1]) designed to capture motion.
- **Audio Content (Speech Recognition):** Convert all speech and dialogue within the video into a text transcript using **PhoWhisper**, a high-accuracy Vietnamese speech recognition model.
- **Specific Object Identification (Object Detection):** To provide a granular list of all recognizable objects within the scene using a **YOLO** model. This complements the general descriptions from captioning by providing specific, itemized data.

Develop an Interactive Chatbot: To build a chatbot that can understand user questions in natural language and use the generated context to find and provide appropriate answers.

Integrate and Test the System: To combine the individual modules (Image Captioning, Video Captioning, PhoWhisper, Object Detection, and Chatbot) into a single, complete system and test its effectiveness through real-world question-and-answer scenarios.

2.3 Proposed Architecture and Solution

To achieve the objectives mentioned above, the proposed system consists of several core components that work together to create a complete information processing workflow.

System Workflow:

When a user uploads a video, the system processes it using four core modules in parallel:

1. **Image Captioning Module:** The system extracts several key frames that represent the overall scenery of the video. These frames are then fed into an Image Captioning model to generate short descriptive sentences. For example, from a video of someone at a gym, this module might generate the description: "*A man is in a gym with various equipment*". The purpose of this module is to summarize the static scene and objects.

2. **Video Captioning Module (based on the BTKG model):** The entire video is analyzed by the Video Captioning model. This model is specifically designed to focus on recognizing motion and actions. The output is one or more sentences describing the main events. For example: "*A man is lifting weights*". This module answers the question, "What is happening?".

3. **Speech Recognition Module (PhoWhisper):** The audio stream from the video is separated and processed by the PhoWhisper model. This model converts all conversations or spoken words in the video into a full text transcript. For instance, if the person in the video says, "*Today, I will try the 100kg weights*", this module will produce that exact text.

4. **Object Detection Module (YOLO):** The same keyframes used for image captioning are also processed by our YOLO model. Instead of a description, this module outputs a specific list of all detected objects. For the gym video, the output might be: `['person', 'dumbbell', 'barbell', 'bench']`. This provides a detailed inventory of what is physically present in the scene.

These four outputs (the static scene description, the action description, the audio transcript, and the object list) are then combined to create a single, comprehensive **text context**. This context acts as the "brain" of the chatbot.

When a user asks a question, the chatbot analyzes the query and searches within this rich text context to provide the most accurate answer.

Illustrative Example:

Consider a video of a person lifting weights. The system would generate a context similar to this:

- **Image Captioning:** "A man wearing a blue shirt and black pants is in a gym."
- **Video Captioning:** "The man is performing an overhead weightlifting press."
- **PhoWhisper Transcript:** "(Sound of breathing) Come on... just one more rep."
- **Object Detection (YOLO):** Detected Objects: `['person', 'dumbbell', 'barbell', 'weight plate', 'bench']`

Based on this context, the chatbot can answer a variety of user questions:

- **User asks:** "What color is the person's shirt?"
  - **Chatbot answers:** "The person in the video is wearing a blue shirt." (Data from Image Captioning)
- **User asks:** "What is that person doing?"
  - **Chatbot answers:** "The person in the video is performing an overhead weightlifting press." (Data from Video Captioning)
- **User asks:** "Is this person saying anything?"
  - **Chatbot answers:** "Yes, the person says: 'Come on... just one more rep'." (Data from PhoWhisper)
- **User asks:** "What equipment can you see in the gym?"
  - **Chatbot answers:** "The equipment I can see includes: a dumbbell, a barbell, a weight plate, and a bench." (Data from Object Detection)

3 Bidirectional Transformer with Knowledge Graph (BTKG)

3.1 Overview

This section describes the proposed Bidirectional Transformer with Knowledge Graph (BTKG) model [1] (Figure 1). We begin by reviewing the basic transformer module and how video features are extracted and captions generated. We then detail the BTKG architecture, including the Spatio-Temporal Encoder, the Objects and Relationships Encoder, and the bidirectional decoder (Backward and Forward Decoders). Finally, we explain the training strategy, including pseudo reverse captions and the loss functions.

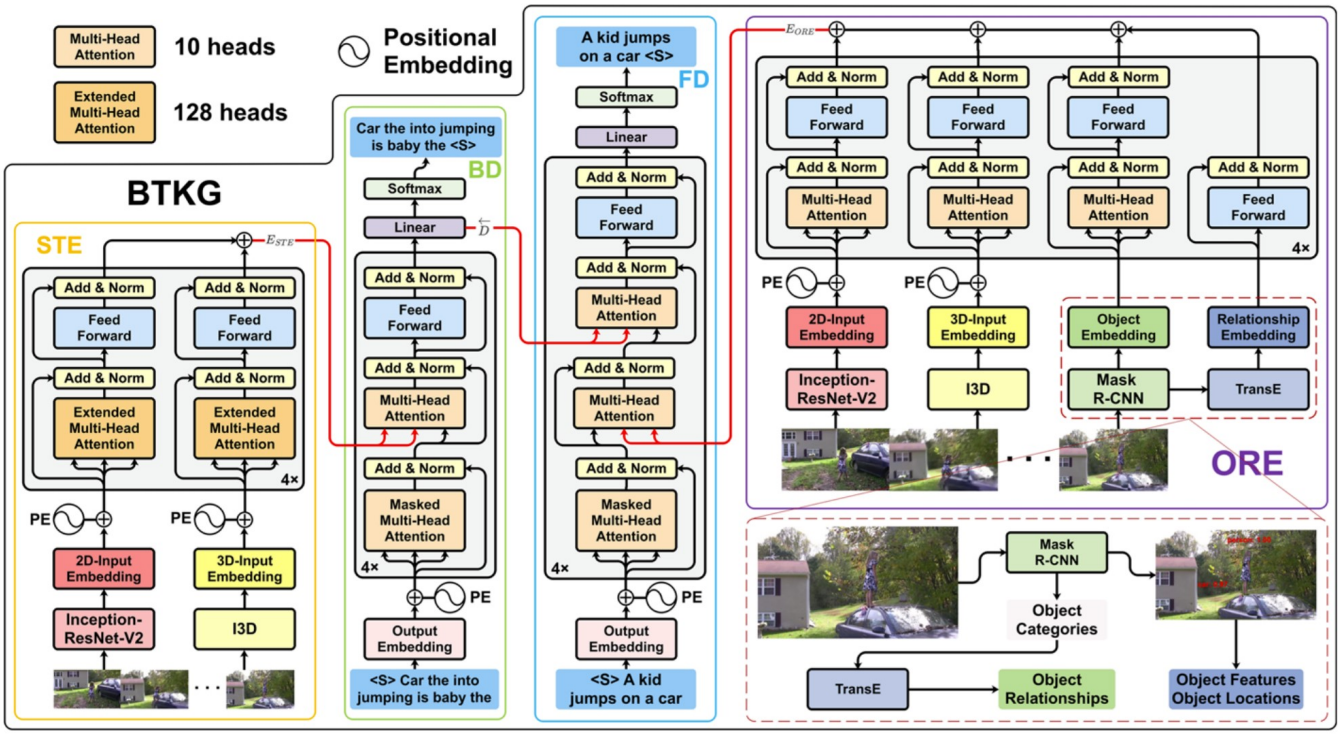

Figure 1: The figure illustrates the proposed BTKG based on the transformer architecture in detail, which consists of Spatio-Temporal Encoder (STE), Objects and Relationships Encoder (ORE), Backward Decoder (BD), and Forward Decoder (FD).

3.2 Basic Module

3.2.1 Transformer Architecture

The model is built upon the standard Transformer [2] architecture. A Transformer consists of an encoder and a decoder, each formed by stacking identical layers. Within each layer are two sub-layers: a multi-head attention (MHA) mechanism and a position-wise feed-forward network (FFN), each followed by residual connections and layer normalization. In the MHA sub-layer, the input queries (Q), keys (K), and values (V) are linearly projected into multiple "heads". Specifically, for head i, we compute QW_i^Q, KW_i^K, and VW_i^V, where W_i^Q, W_i^K, W_i^V are learned projection matrices.

The attention outputs of the h heads are concatenated and projected via W^O:

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\, W^O, \\ \text{head}_i &= \text{Attention}(QW_i^Q,\, KW_i^K,\, VW_i^V). \end{aligned} \tag{1}$$

The Attention function uses scaled dot-product attention:

$$\text{Attention}(Q_i, K_i, V_i) \;=\; \text{softmax}\Big(\frac{Q_i K_i^T}{\sqrt{d_k}}\Big)\, V_i,$$

where d_k is the dimensionality of the key vectors.

The FFN sub-layer applies two linear transformations with a ReLU nonlinearity in between. For an input X,

$$\text{FFN}(X) = \max(0,\, XW_1 + b_1)\, W_2 + b_2,$$

with parameters W_1, W_2, b_1, b_2. Together, these MHA and FFN components provide the basic sequence modeling capability of the Transformer.

3.2.2 Multimodal Features Extraction and Captions Generation

The task is video captioning, so the model must encode both visual and dynamic information. To this end, the system extracts **image features** and **motion features** from the video. It uses the Inception-ResNet-V2 network (pretrained on ImageNet) to extract per-frame image features, capturing static appearance. For motion, it uses the Inflated 3D ConvNet (I3D) pretrained on the Kinetics dataset to extract features from sequences of frames (capturing temporal dynamics). These networks produce feature sequences $F_I = \{f_1, \ldots, f_L\}$ and $F_M = \{v_1, \ldots, v_N\}$, where $f_i \in \mathbb{R}^{d_I}$ are image features for L sampled frames and $v_j \in \mathbb{R}^{d_M}$ are motion features (with $N = L/64$ in practice). Both feature sequences are then projected into a common model dimension d_{model} via learned linear mappings.

In addition to these frame-level features, the model incorporates **object-level semantics**. A Mask R-CNN detector (trained on the COCO dataset) detects objects in each frame and produces object feature vectors and class labels. For each detected object, its class name (e.g. "car", "person") is used in a knowledge graph module: a TransE-based model predicts relationships (triplets) between object categories. The object labels and their predicted relations are embedded via a shared word-embedding matrix (e.g. Word2Vec) to produce relation features.

For caption generation, this base module uses a bidirectional decoding strategy [3]: there is a backward decoder and a forward decoder. The backward decoder generates captions right-to-left (reverse order), while the forward decoder generates left-to-right. This bidirectional decoder takes as input the encoded video features. The backward decoder's output (a reverse caption and its hidden context) is fed into the forward decoder to improve forward captioning. In summary, the basic module provides: a Transformer backbone, multimodal feature encodings (image, motion, objects, relations), and a bidirectional decoding scheme.

3.3 BTKG Architecture

The BTKG model integrates the above components into a unified architecture. It connects a bidirectional decoder with an encoder that incorporates knowledge-graph-enriched object features. As shown in Figure 1, BTKG consists of four parts:

- **Spatio-Temporal Encoder (STE):** encodes fine-grained frame and motion features to produce E_{STE}.
- **Objects and Relationships Encoder (ORE):** encodes object features and their relationships (along with coarse video features) to produce E_{ORE}.
- **Backward Decoder (BD):** generates a reverse caption Y_{BD} (right-to-left) and outputs a hidden context $\overleftarrow{D}$.
- **Forward Decoder (FD):** generates the final caption Y_{FD} (left-to-right), integrating both E_{ORE} and the reverse context $\overleftarrow{D}$.

The next subsections explain each component in detail.

3.3.1 Spatio-Temporal Encoder (STE)

The STE is designed to capture the fine-grained spatio-temporal content of the video. It operates on the image and motion feature sequences extracted from the video frames (as described above). Formally, let $X = \{x_1, \ldots, x_L\}$ be the sampled frames. We extract image features

$$F_I = \{f_1, \ldots, f_L\}, \quad f_i \in \mathbb{R}^{d_I},$$

and motion features

$$F_M = \{v_1, \ldots, v_N\}, \quad v_j \in \mathbb{R}^{d_M}, \quad N = L/64,$$

using Inception-ResNet-V2 and I3D respectively. To combine these modalities, each feature vector is linearly projected to a common dimension d_{model} by learned weights:

$$f'_i = W_L f_i + b_L, \quad v'_j = W_N v_j + b_N,$$

where $W_L \in \mathbb{R}^{d_{\text{model}} \times d_I}$, $W_N \in \mathbb{R}^{d_{\text{model}} \times d_M}$, and biases $b_L, b_N \in \mathbb{R}^{d_{\text{model}}}$. This yields transformed sequences $F'_I = \{f'_1, \ldots, f'_L\}$ and $F'_M = \{v'_1, \ldots, v'_N\}$, where $f'_i, v'_j \in \mathbb{R}^{d_{\text{model}}}$.

The sequences F'_I and F'_M are then each fed through a Transformer encoder (with self-attention and FFN sublayers) to produce embeddings $E_I \in \mathbb{R}^{L \times d_{\text{model}}}$ and $E_M \in \mathbb{R}^{N \times d_{\text{model}}}$. Finally, these are fused by element-wise addition to produce the STE output:

$$E_{STE} = E_I \oplus E_M,$$

where $\oplus$ denotes element-wise sum. Thus $E_{STE} \in \mathbb{R}^{L \times d_{\text{model}}}$ encodes frame-level (fine-grained) video content by blending appearance and motion features.

A notable detail is that in the STE, the model uses an **extended multi-head attention** with 128 heads (instead of the usual 8). Using more heads allows the model to capture correlations among positions at a finer temporal scale than whole frames. In practice, the STE employs $h = 128$ attention heads as indicated in Eq. 1, which helps BTKG capture fine temporal dependencies.

3.3.2 Objects and Relationships Encoder (ORE)

The ORE captures higher-level semantics by encoding detected objects and their relationships. First, objects are detected in the video using Mask R-CNN (with a Feature Pyramid Network and ROI pooling). For each detected object i (up to k objects), we record its normalized bounding box coordinates and a visual feature vector. Let $R_l = [l_1, \ldots, l_k]$ be the list of object locations (each l_i is 4-dimensional) and $R_v = [v_1, \ldots, v_k]$ the list of object feature vectors. These are concatenated per-object: the i-th object feature is $(l_i|v_i)$, and stacking these forms the object feature matrix F_o:

$$F_o = \text{concat}_{i=1}^{k}\big(l_i,\, v_i\big).$$

in the paper's notation this is written as

$$F_o = \text{concat}_{i=1,\ldots,k}(R_l\, R_v),$$

meaning the row-wise concatenation of R_l and R_v for each object.

In parallel, the object class labels (one-hot vectors o_i) are input to the TransE knowledge graph model to predict relations between objects. This yields a set of relation one-hot vectors $\{r_1, \ldots, r_m\}$, where $m = \binom{k}{2}$. Both the object labels o_i and relation labels r_j are embedded into the same vector space via a shared embedding matrix W (pretrained by Word2Vec). Specifically:

$$o_i' = o_i W, \quad r_j' = r_j W,$$

producing embedded matrices $R_o' = [o_1', \ldots, o_k']$ and $R_r' = [r_1', \ldots, r_m']$. These are concatenated vertically to form the relation feature matrix F_r:

$$F_r = \begin{bmatrix} R_o' \\ R_r' \end{bmatrix}$$

The matrix F_r thus contains semantic features of objects and predicted relations.

Next, the ORE encodes the content of F_I', F_M' (the same projected image/motion features as STE), F_o, and F_r. As in the STE, F_I' and F_M' are passed through transformer encoders (here using 10 attention heads) to yield embeddings E_I and E_M. Then the object feature matrix F_o is encoded (via a transformer encoder without positional encoding, since object order is arbitrary) to produce an object embedding E_o. Likewise, the relation matrix F_r is encoded by a simple feed-forward network (no attention, since relations are unordered) to produce a relation embedding E_r.

Finally, these four embeddings are fused by element-wise addition:

$$E_{ORE} = E_I \oplus E_M \oplus E_o \oplus E_r.$$

This sum is the output of the ORE. Thus $E_{ORE} \in \mathbb{R}^{L \times d_{\text{model}}}$ encodes coarse video features (from E_I, E_M) enriched by object and relation semantics (E_o, E_r).

3.3.3 Backward Decoder (BD)

The Backward Decoder generates a caption in reverse order (right-to-left). It takes the STE output E_{STE} as its encoder memory; it does not use E_{ORE}, because object relations have no natural reverse-time ordering and could confuse the backward generation. The BD is implemented as a transformer decoder: at each step it attends to E_{STE} and generates one word from the end of sentence toward the beginning. Generation stops when the end marker $< S >$ is produced. If the generated reverse caption is $\overleftarrow{C} = [s_1, s_2, \ldots, s_L, < S >]$, then the final hidden state of the BD (after producing s_L) is denoted $\overleftarrow{D}_L$. We define the BD's context output as

$$\overleftarrow{D} = \overleftarrow{D}_L$$

This vector $\overleftarrow{D}$ summarizes the reverse caption's context and is passed on to the forward decoder.

3.3.4 Forward Decoder (FD)

The Forward Decoder generates the final caption left-to-right. It integrates two sources of context: the video encoding and the backward decoder's context. Concretely, the FD attends to E_{ORE} (via a standard encoder-decoder attention) so that object and relation features inform each predicted word. It also attends to the reverse caption context $\overleftarrow{D}$ via an extra cross-attention layer. In practice, at each step the FD takes as input all previously generated words and uses cross-attention over $\overleftarrow{D}$ so that it "sees" the whole reverse caption context. It similarly attends to E_{ORE} (which contains object/relation/video info). Generation ends at $< S >$, producing a forward caption $\overrightarrow{C} = [s_1, \ldots, s_T, < S >]$. Thus, the FD effectively conditions each word on both E_{ORE} and $\overleftarrow{D}$.

3.4 Proposed Enhancements

3.4.1 Learnable Multimodal Feature Fusion

Background and Limitation

In the original BTKG architecture, multimodal features are combined using element-wise addition. Specifically, the Spatio-Temporal Encoder (STE) fuses image and motion features via the formula:

$$E_{STE} = E_I \oplus E_M$$

Similarly, the Objects and Relationships Encoder (ORE) combines all four feature types as:

$$E_{ORE} = E_I \oplus E_M \oplus E_o \oplus E_r$$

This method has an inherent limitation: it assumes that each modality contributes equally to the final representation. Element-wise addition is a static, non-learnable operation that cannot dynamically adjust the weight of each feature type based on the video's context.

Proposed Solution: Feature Fusion Module

To overcome this drawback, we replace the element-wise addition with a learnable **Feature Fusion** module. This module employs a Convolutional layer with a kernel size of 1, which effectively acts as a shared linear layer to intelligently combine features.

The **Feature Fusion** module operates as follows:

1. **Concatenation**: Instead of adding them, the feature tensors from different modalities (e.g., E_I, E_M, E_o, E_r), which share the shape `(batch_size, seq_len, d_model)`, are concatenated. This creates a new tensor of shape `(batch_size, n_features, seq_len, d_model)`, where `n_features` is the number of modalities.
2. **1x1 Convolution**: The concatenated tensor is then passed through a `Conv` layer with a kernel size of 1. This convolutional layer serves as a learnable linear projection. In essence, it learns how to "mix" the concatenated features to produce a new, more concise, and meaningful representation.
3. **Output Reshaping**: The output tensor from the `Conv` layer has the shape `(batch_size, seq_len, d_model)`, making it fully compatible with the rest of the Transformer architecture and a direct replacement for the previous element-wise sum.

Advantages of this Improvement:

- **Learnability**: The weights of the `Conv` layer are updated during training. This allows the model to automatically learn the relative importance of each feature type. For instance, in a video with complex actions, the model might learn to assign higher weights to motion features (E_M) and relationship features (E_r).
- **Enhanced Representation Power**: By allowing features to interact through a learned linear transformation, the model can create a richer and more expressive joint representation space compared to a simple summation.

3.4.2 Improving the Transformer Architecture with ResiDual Connections

Background and Problem

The Transformer architecture in the original BTKG model utilizes **Post-Layer Normalization (Post-LN)**. In this structure, a residual connection is performed by adding the input and output of a sub-block (e.g., Multi-Head Attention), after which Layer Normalization is applied.

While common, the Post-LN architecture suffers from critical limitations, especially when building deep models:

1. **Gradient Vanishing**: As the gradient signal backpropagates from the top layers to the bottom ones, its magnitude decays exponentially. This is because the gradient must pass through numerous Layer Normalization operations, which hinders the effective training of deeper layers.
2. **Training Instability**: Successfully training deep Post-LN Transformers often requires auxiliary techniques like learning-rate warm-up to mitigate instability in the early stages of training.

Proposed Solution: Adopting the ResiDual Architecture

To address these issues and enhance model stability and performance, we replaced the original Post-LN architecture with **ResiDual**, an advanced architecture proposed by Xie et al. (2023) [4] at Microsoft. ResiDual is designed to combine the advantages of both Post-LN and Pre-Layer Normalization (Pre-LN) while eliminating their respective drawbacks.

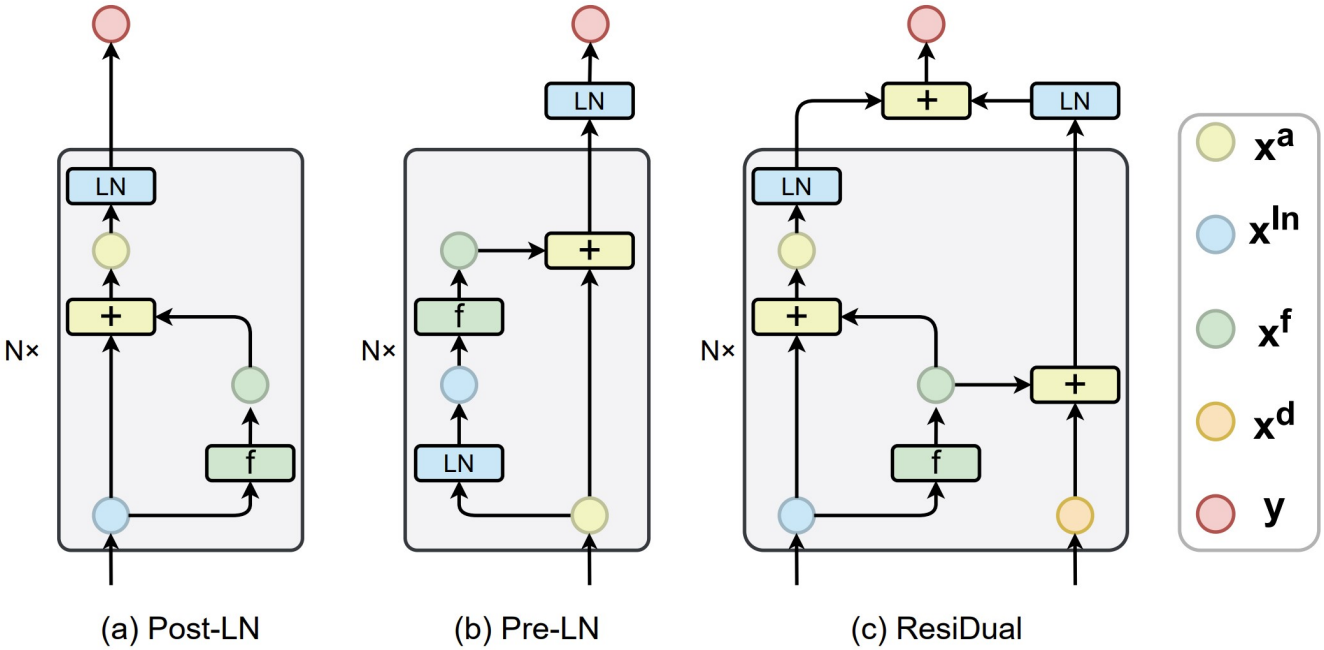

Figure 2: Overview of Post-LN, Pre-LN, and ResiDual. Circles with different colors represent different variables and rectangles represent different operations.

The ResiDual architecture utilizes **two parallel residual branches**:

- **Main Branch (Post-LN-like)**: This branch retains the structure of Post-LN, where the sub-block's output is added to its input and then normalized. This branch is crucial for **maintaining representation diversity**, which helps combat the "representation collapse" phenomenon often observed in Pre-LN.
- **Dual Branch (Pre-LN-like)**: A second residual path is added, allowing the signal (both forward and backward) to bypass the blocks. This branch accumulates the outputs of the sub-blocks and allows the gradient to flow directly to the deepest layers without being attenuated by normalization layers. This effectively **solves the gradient vanishing problem**.

Benefits of Adopting ResiDual:

- **Stable Training and Better Convergence**: By providing an unimpeded path for the gradient, ResiDual makes the training process significantly more stable, especially as model depth increases. This reduces the dependency on techniques like learning-rate warm-up.
- **Enhanced Representation Capacity**: The architecture prevents representation collapse, ensuring that deeper layers can continue to effectively learn and refine features. This allows

the model to fully leverage its parameters, leading to a more powerful and robust model.

3.5 Training

BTKG is trained end-to-end with cross-entropy losses on both decoders. The training involves a two-stage process: first use the backward decoder to generate (pseudo) reverse captions, then use those in the forward decoder. Let D be the training set of videos with forward ground-truth captions. For each video V with forward caption $\overrightarrow{Y} = [y_1, \ldots, y_L]$, we first obtain a pseudo reverse caption (described below) and train the BD on that. The FD is then trained on the forward caption, conditioned on the BD's output. Denote the BD loss by L_{bd} and the FD loss by L_{fd} (both standard token-level cross-entropy). The total training loss is a weighted sum:

$$L = (1 - \lambda)\, L_{bd} + \lambda\, L_{fd},$$

where $\lambda \in [0, 1]$ balances the two parts.

3.5.1 Pseudo Reverse Captions

To avoid trivial information leakage, the reverse captions used for training are pseudo. All ground-truth forward captions for a video are reversed in order (word sequence flipped) to create candidate reverse captions of the same length. Then these reversed captions are randomly shuffled and paired with the original video, so the final training reverse caption is not exactly the true reverse of its forward caption. In other words, for training we do not feed the exact future ground-truth to the BD. This randomization mitigates the leakage issue and prevents the model from cheating by memorizing exact reversals.

3.5.2 Backward Decoder Loss

The BD is trained to minimize the negative log-likelihood of the pseudo reverse captions. Formally, if $(V, \overleftarrow{Y}) \in D$ where $\overleftarrow{Y}$ is a training reverse caption (of length T), then

$$L_{bd} = \sum_{(V, \overleftarrow{Y}) \in D} \sum_{t=1}^{T} -\log p(y_t \mid V,\, y_1, \ldots, y_{t-1};\, \theta_{ste}, \theta_{bd}),$$

where y_t is the t-th token in $\overleftarrow{Y}$, and $\theta_{ste}, \theta_{bd}$ are the STE and BD parameters. This is the standard cross-entropy loss for the backward decoder.

3.5.3 Forward Decoder Loss

Similarly, the FD is trained by cross-entropy on the forward captions. Importantly, each forward token probability is conditioned on both the video and the reverse-context $\overleftarrow{D}$. The loss is:

$$L_{fd} = \sum_{(V, \overrightarrow{Y}) \in D} \sum_{l=1}^{L} -\log p(y_l \mid V,\, y_1, \ldots, y_{l-1};\, \theta_{ste}, \theta_{bd}, \theta_{ore}, \theta_{fd}),$$

with $\theta_{fd}, \theta_{ore}$ the FD and ORE parameters. Though the form looks like a normal forward loss, recall that in the FD decoder architecture each step attends to $\overleftarrow{D}$ (the BD context), allowing the model to use "future" information encoded by the BD.

In summary, training optimizes $L = (1 - \lambda)L_{bd} + \lambda L_{fd}$ so that the BD learns to generate accurate reverse captions and the FD learns to generate accurate forward captions given both video features and the reverse-context. This bidirectional setup with pseudo-reverse captions is designed to reduce information leakage and fully exploit the video semantics captured by the Knowledge Graph encoder.

4 Keyframe-Based Image Captioning

Beyond the primary video-level architecture, we further investigate a complementary paradigm that treats video captioning as a sequence of keyframe-driven image-captioning tasks. In this variant, keyframes are first distilled from the clip and each is captioned by a image-captioning model, rather than processing the entire video sequence.

4.1 Keyframe Selection

The keyframe selection procedure is designed to minimize redundancy and extract only those frames that represent significant visual changes within the video. This process involves the following steps:

1. Frame Sampling: Frames are initially sampled from the video at a rate of 1 frame per second (fps). This sampling rate substantially reduces the processing load while maintaining a linear runtime relative to the video duration, which is well-suited for short, single-activity videos typical of the MSVD dataset.
2. Feature Extraction via Visual Embedding: Each sampled frame is transformed into a high-dimensional feature vector using a Vision Transformer (google/vit-base-patch16-224-in21k [5]) model pre-trained on the ImageNet-21k dataset. ViT embeddings were selected for their demonstrated robustness to common visual artifacts such as motion blur and variations in illumination, rendering them a reliable foundation for change detection.
3. Similarity-Based Filtering: A frame is designated as a keyframe if its visual representation is sufficiently dissimilar from that of the preceding keyframe. This determination is made using a cosine similarity metric. A frame is added to the keyframe set if the cosine similarity score between its embedding and that of the last accepted keyframe is below a threshold of 0.95.

4.2 Image Captioning Model

4.2.1 Model Overview

The image captioning model employs a sophisticated encoder-decoder architecture, integrating a pre-trained Vision Transformer (ViT) [6] as the encoder and a pre-trained Transformer-based language model (T5) [7] as the decoder.

The visual encoder is a ViT-Base/16 [8] model, pre-trained on the ImageNet-21k dataset. Its function is to process an input image and generate a sequence of patch embeddings, which serve as a rich, high-dimensional representation of the image's content.

The text decoder is based on the VietAI/vit5-base [9] model, a T5 variant specifically pre-trained for the Vietnamese language. To bridge the modality gap between the visual encoder and the text decoder, a dedicated interface mechanism was implemented. A linear projection layer maps the embedding dimension of the ViT's output features to the expected hidden dimension (d_{model}) of the T5 decoder.

During the forward pass, the projected visual features directly substitute the output of the T5's own text encoder. Consequently, the T5 decoder's cross-attention mechanism attends directly to this sequence of visual embeddings, enabling it to generate a textual description conditioned on the

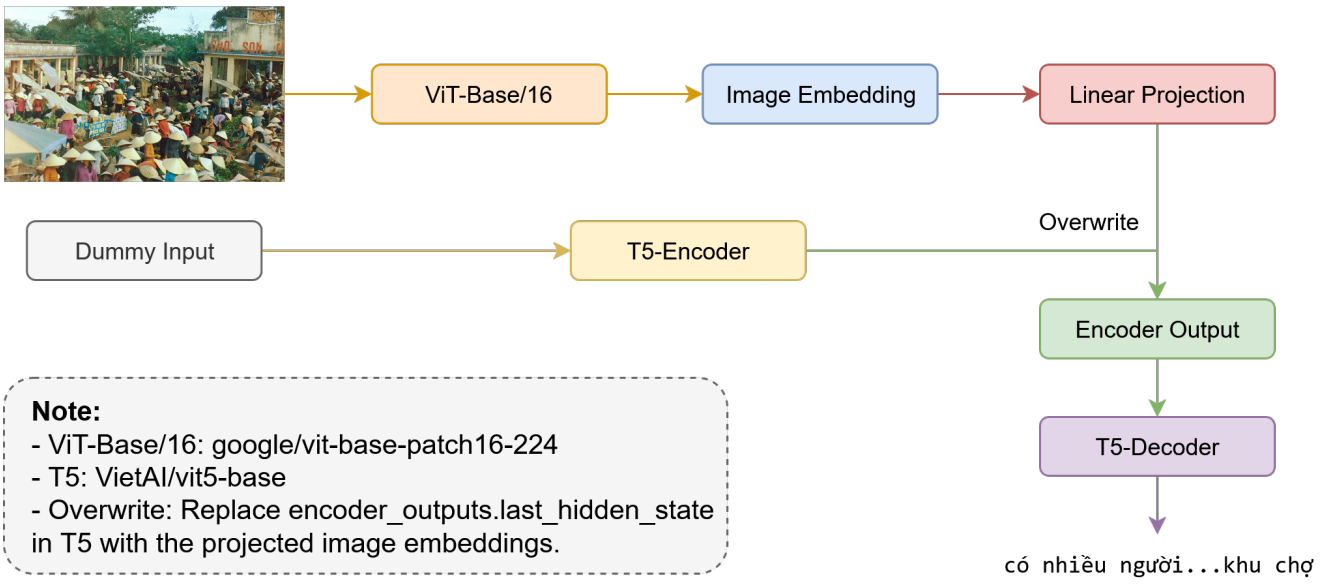

Figure 3: The architecture of the ViT-T5 image captioning model, illustrating the integration of the Vision Transformer (ViT) encoder and the T5 decoder.

image content. For generation, the model utilizes a beam search with a beam width of 3 to produce fluent and coherent captions.

4.2.2 Encoder: Vision Transformer (ViT)

The choice of the Vision Transformer (ViT) as the encoder was driven by its fundamental architectural advantages over traditional Convolutional Neural Networks (CNNs). Unlike the local receptive fields of CNNs, ViT's self-attention mechanism allows it to capture global context by modeling long-range dependencies between all image patches simultaneously. This holistic understanding of the scene is critical for generating accurate and coherent captions that describe complex interactions between objects. Guided by this motivation, we adopt a pre-trained ViT-Base/16 as our visual encoder.

Concretely, the visual encoder is the backbone of our image captioning model, responsible for transforming a raw image into a sequence of rich feature embeddings that the text decoder can interpret. We employ a pre-trained ViT-Base/16 model, which processes the image as follows (see Figure 4):

- **Image Patching:** First, instead of processing pixels individually, the ViT architecture treats an image as a sequence. The input image is divided into a grid of fixed-size, non-overlapping patches. For ViT-Base/16, this means patches of size 16×16 pixels.
- **Linear Projection of Patches:** Next, each patch is flattened into a one-dimensional vector and linearly projected into a fixed-dimensional embedding space, yielding a sequence of "patch embeddings"—analogous to word embeddings in NLP.
- **Positional Embeddings:** Then, because the Transformer is permutation-invariant, learn-

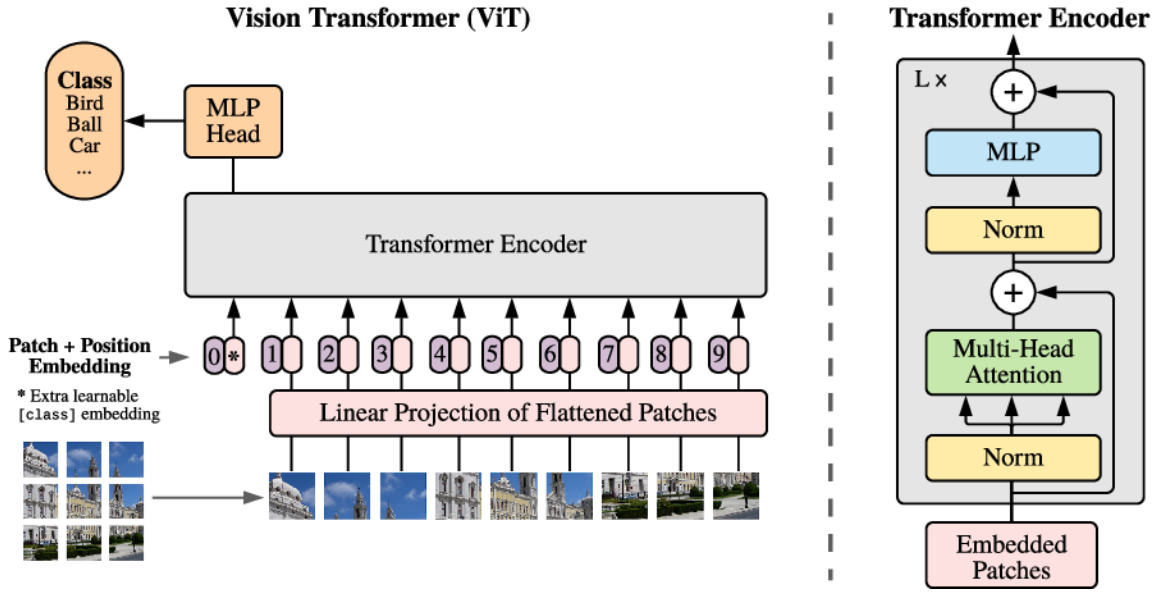

Figure 4: The architecture of the Vision Transformer model.

able positional embeddings are added to provide spatial context, enabling the model to reason about relative locations of patches.

- **Transformer Encoder Layers:** Finally, the resulting sequence is fed into a standard Transformer encoder with:
  - *Multi-Head Self-Attention (MHSA):* to weigh relationships among patches (e.g., "person" near "bicycle").
  - *Feed-Forward Network (FFN):* a position-wise MLP that further processes each embedding.

The encoder outputs a sequence of contextually-aware patch embeddings, which serve as a high-level representation of the image and are subsequently consumed by the T5 decoder to generate the textual description.

4.2.3 Decoder: T5 Language Model

The text decoder is the component responsible for generating the final descriptive sentence. For this role, we select VietAI/vit5-base [9], a T5 variant pre-trained on a large corpus of Vietnamese text. This choice leverages its strong modeling of Vietnamese grammar, syntax, and semantics to produce fluent and natural captions.

While a standard T5 model comprises both an encoder and a decoder, our architecture exclusively utilizes the decoder for text generation. The core challenge is bridging the modality gap so that the text-based T5 decoder can condition on the visual features produced by the ViT encoder. To this end, we replace T5's text encoder with the ViT-derived features via the following steps:

1. **Dimension alignment via linear projection.** Let the ViT patch embeddings have dimension d_{ViT} (for ViT-Base/16, d_{ViT}=768). The T5 decoder expects inputs of hidden size

d_{model} (for `vit5-base`, d_{model}=768). We pass the ViT outputs through a learnable linear layer $W \in \mathbb{R}^{d_{\text{ViT}} \times d_{\text{model}}}$ to map visual features into the T5 latent space with minimal information loss.

2. **Encoder-output substitution.** Instead of using T5's original text encoder, the projected visual features are used to *overwrite* the encoder outputs. Concretely, we construct an encoder output object whose `last_hidden_state` equals the sequence of projected patch embeddings $\mathbf{H} \in \mathbb{R}^{N \times d_{\text{model}}}$, where N is the number of visual tokens.

3. **Cross-attention over visual features.** The T5 decoder employs cross-attention at each generation step to attend to the encoder outputs. By substituting the encoder outputs with $\mathbf{H}$, the decoder directly attends to the image's visual representation, enabling token generation to be conditioned on salient regions of the image.

This integration combines the ViT's strong visual representations with T5's robust text generation, yielding an effective image-captioning pipeline in Vietnamese.

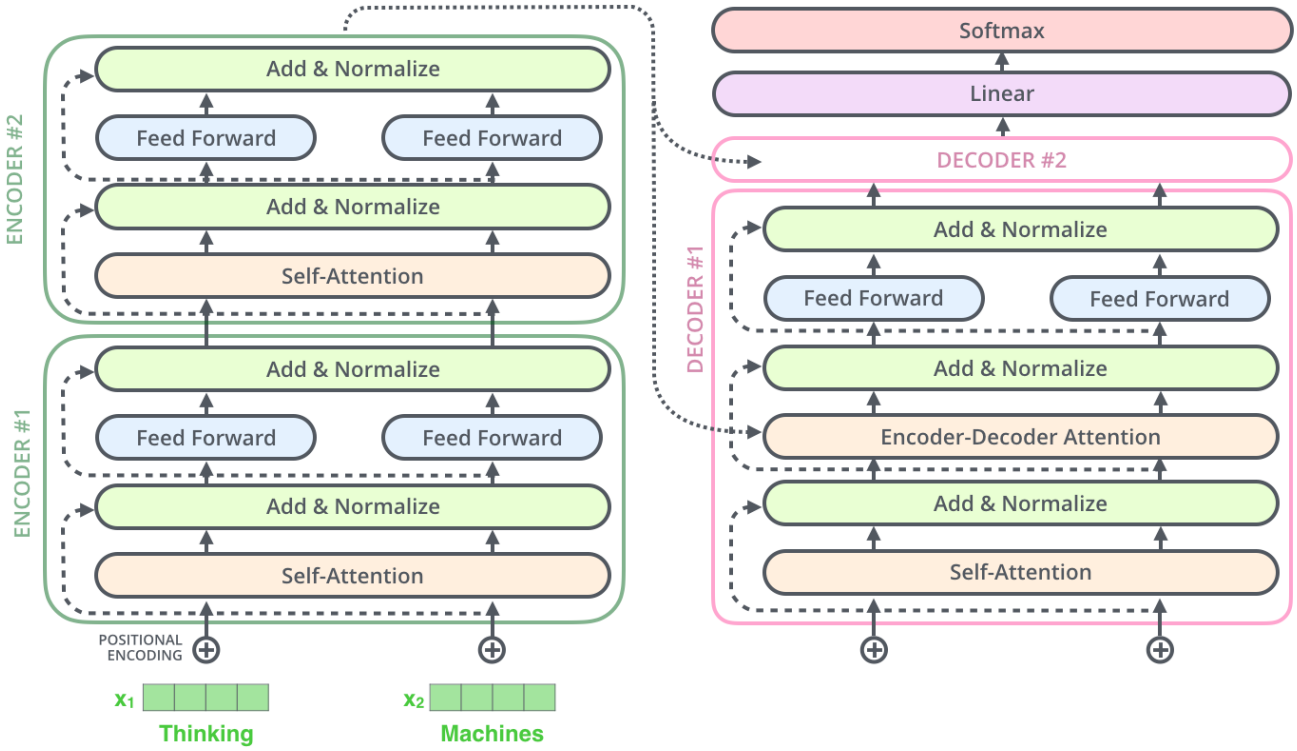

Figure 5: The architecture of the T5 model.

4.2.4 Training Procedure

The model was trained using a parameter-efficient fine-tuning (PEFT) strategy to leverage the knowledge from the pre-trained components while minimizing computational cost and the risk of catastrophic forgetting. During this phase, the weights of the entire ViT encoder and the T5 text

encoder were frozen and did not receive updates. Only the parameters of the T5 decoder, the final language model head, and the intermediary linear projection layer were made trainable.

The model was optimized by minimizing the standard cross-entropy loss between the predicted token sequence and the ground-truth captions. We employed the AdamW optimizer with an initial learning rate of 5×10^{-5} and a weight decay of 1×10^{-4}. The learning rate was dynamically adjusted throughout training using a Cosine Annealing schedule with warm restarts.

Training was conducted for a maximum of 8 epochs. To prevent overfitting and select the best-performing model, an early stopping criterion was implemented. This mechanism monitored the validation loss at the end of each epoch and terminated the training process if no improvement was observed for 5 consecutive epochs. The model checkpoint that achieved the lowest validation loss was saved and used for all subsequent evaluations and inference tasks.

5 Experiments

5.1 Experimental Settings

5.1.1 Datasets

The MSVD dataset (Microsoft Research Video Description) was used as the evaluation benchmark for BTKG model. MSVD contains 1,970 short YouTube video clips (10-25 seconds each) with roughly 80,000 English sentences (about 40 captions per clip). These videos are split following standard practice: 1,200 clips for training, 100 for validation, and 670 for testing. (All captions are in English and describe a single-activity scene per clip, as in prior work.)

For the image captioning model, we used the COCO-2017 dataset with Vietnamese captions [10], and for Vietnamese video captioning we used the KTVIC dataset [11]. The training split contains 619,723 captions aligned with 123,879 images (appropriate 5 captions per image). The validation split has 3,177 images with one caption each (3,177 captions), and the test split has 558 images with one caption each (selected from the original KTVIC test list).

5.1.2 Feature extraction (BTKG)

We extract both spatial and motion features from the video clips. Video frames were sampled at 5 frames per second (fps) and fed into a pretrained Inception-ResNet-V2 [12] network to obtain image features. Concurrently, videos were resampled at 25 fps and divided into overlapping 64-frame segments (stride of 5 frames); these segments were processed by an Inflated 3D ConvNet (I3D) [13] to compute motion features. The resulting image and motion feature vectors have dimensions *2048* and *1024*, respectively. We also encode each video's category label using a 300-dimensional GloVe word vector. From each video's features, 50 frames are uniformly sampled, and the corresponding image, motion, and category feature vectors are then projected into a shared 512-dimensional space.

5.1.3 Evaluation metrics

We evaluate both tasks–(i) video captioning on MSVD and (ii) keyframe-based image captioning on the Vietnamese datasets–using the same suite of automatic metrics: BLEU [14], METEOR [15], CIDEr-D [16], and ROUGE-L [17]. BLEU and METEOR (originating from machine translation) quantify n-gram overlap between generated and reference captions. CIDEr-D, designed for caption evaluation, measures consensus with human annotations, while ROUGE-L (based on longest common subsequence) emphasizes recall. Unless otherwise noted, we report corpus-level scores; BLEU refers to BLEU@4, CIDEr to the CIDEr-D variant, and ROUGE to ROUGE-L.

5.1.4 Parameter settings

We detail the model's settings for MSVD experiments. The object detector Mask R-CNN [18] used a confidence threshold of 0.7 and minimum box size of 224×224. A TransE knowledge-graph was built from 613 unique triples, with each relationship encoded by a 300-dimensional GloVe vector [19]. The Transformer encoders and decoders use 4 layers, with 640-dimensional input embeddings, 512-dimensional model hidden states, 2048-dimensional feedforward layers, and 10 attention heads per

layer. (An extended multi-head attention in the STE component uses 128 heads.) During training on MSVD, the balance weight λ between forward/backward decoding was set to 0.6 and the model was trained for 30 epochs. We used the Adam optimizer with a batch size of 32 and a learning rate of 1e−4 for MSVD training. Inference used beam search of size 5 with dropout 0.1.

For the image captioning model, we train a ViT–T5 captioner that uses a ViT-Base/16 vision encoder (vit_base_patch16_224) and a Vietnamese T5 decoder (VietAI/vit5-base). Images are resized to 224×224 and normalized with ImageNet mean/std; a linear layer projects ViT patch embeddings to the T5 d_{model}, whose encoder hidden states are replaced by the projected visual features. Captions are tokenized with the T5 tokenizer and generated with a maximum length of 35 tokens. Training uses AdamW (learning rate 5e−5, weight decay 1e−4), batch size 32, cosine-annealing with warm restarts (T_0=1000, $\eta_{\min}$=1e−7), and early stopping with patience 5; we train for 8 epochs. Data loading uses pinned memory and `num_workers` equal to the machine's CPU cores. Inference uses beam search with beam size 3 and early stopping.

All experiments were run on NVIDIA Tesla P100 GPUs.

5.2 Results and Discussion

5.2.1 BTKG Model

We evaluated our proposed enhancements on the MSVD dataset. The table below presents a comparison between the results published in the original paper, our re-implemented baseline, and our enhanced models.

Model Version	**BLEU@4**	**METEOR**	**ROUGE-L**	**CIDEr-D**
Published Results	55.7	38.3	74.7	104.5
BTKG (Our Baseline)	54.6	37.9	74.3	96.8
BTKG + Feature Fusion	55.1	38.0	74.1	98.1
BTKG + Feature Fusion + ResiDual	55.5	38.3	74.9	100.2

Table 1: Performance comparison of BTKG models on MSVD dataset.

Comparison with Published Baseline: Our re-implementation of the original BTKG model yielded slightly lower scores than those reported in the paper, most notably a drop in the CIDEr-D score from 104.5 to 96.8. This discrepancy can be attributed to differences in the experimental environment (the original paper used an RTX 3060 GPU, whereas our experiments were run on an NVIDIA Tesla P100 on Kaggle) or minor variations in data preprocessing. Nonetheless, this result serves as a solid baseline for evaluating our proposed improvements.

Effectiveness of the Feature Fusion Module: Replacing element-wise addition with our learnable **Feature Fusion** module led to a noticeable improvement across most metrics, especially CIDEr-D, which increased from 96.8 to 98.1. This confirms that a learnable, weighted combination

of features is more effective than a simple, static summation. The model can better prioritize more relevant information, resulting in a richer fused feature vector for the decoder.

Positive Impact of the ResiDual Architecture: The most significant improvement came from integrating the **ResiDual** architecture. The "**BTKG + Feature Fusion + ResiDual**" model substantially outperformed our baseline across all metrics. Notably, the CIDEr-D score saw a strong increase to 100.2, and the METEOR score reached 38.3, matching the published result. By stabilizing the training process and preventing representation collapse, ResiDual allows the deeper layers of the model to learn more complex and diverse representations. This directly enhances the model's ability to capture the fine-grained semantics of the video, leading to more accurate and semantically appropriate captions, as reflected by the significant gain in the CIDEr-D score.

Observations

While our best-performing model has not yet surpassed all the scores from the original publication, our experimental results clearly demonstrate that:

1. Using a learnable mechanism like **Feature Fusion** is more effective for combining multimodal features than direct summation.
2. Upgrading the Transformer architecture to a more stable and powerful design like **ResiDual** yields significant performance gains, particularly in the semantic quality of the generated captions.

These findings validate our research direction and show that the proposed enhancements have great potential for improving the overall effectiveness of the BTKG model for video captioning.

5.2.2 ViT-T5 Image Captioning Model

We also evaluated the ViT-T5 image captioning model on the KTVIC dataset, which contains images with Vietnamese captions.

Encoder - Decoder	BLEU@4	METEOR	ROUGE-L	CIDEr-D
ResNet101 - LSTM	15.5	27.5	42.3	21.8
ViT - Transformer	19.3	30.1	44.7	36.8
GRIT - GRIT	**40.6**	36.6	**59.7**	**136.0**
ViT - T5 (Ours)	30.3	**48.8**	57.3	88.3

Table 2: Performance of our ViT-T5 model on the KTVIC dataset. Results for ResNet101-LSTM, ViT-Transformer, and GRIT-GRIT are reported from the original KTVIC dataset paper [11] for reference.

These results highlight an effective trade-off between performance and architectural simplicity. While the more complex GRIT model sets a high benchmark, our ViT-T5 model, which is based on a standard Transformer architecture appropriate for the scope of this course, proves to be

highly effective. It not only surpasses the ResNet101-LSTM and ViT-Transformer baselines on all metrics but also achieves the highest METEOR score, indicating its particular strength in capturing semantic similarity and fluency.

6 Applications

We investigated the applications of our model across various domains. The model's ability to generate coherent and contextually relevant text makes it suitable for question-answering tasks with a chatbot.

6.1 Chatbot Application

We used Gemini 2.5 Flash [20] to create a chatbot that can answer questions based on the provided context. The chatbot is designed to understand and respond to user queries in a conversational manner, leveraging the model's capabilities to generate human-like responses.

The system operates in a two-stage process: context generation followed by user interaction. The first stage is initiated upon video upload. The system automatically subjects the video to a multi-modal analysis pipeline involving four concurrent modules: a video captioning model for action summaries, a keyframe captioning model for scene details, an object detection model for itemized lists, and a speech recognition model for audio transcription. The information from these sources is aggregated into a unified contextual document. In the second stage, the user can query the chatbot, which operates exclusively on this pre-generated context to deliver precise answers without re-processing the video.

6.2 Video Captioning Implementation

To generate a dynamic, event-based understanding of the video, we first partition the input video into continuous 20-second segments. Each segment is then fed into the video captioning model, which produces a concise description of the main actions and events occurring within that timeframe. The resulting sequence of captions provides a narrative summary that serves as the primary context for the chatbot.

6.3 Keyframe Captioning Implementation

To complement the action-based summaries, we implement keyframe captioning to capture static, high-fidelity visual details. The pipeline begins by extracting keyframes from the video, which are frames chosen to represent distinct visual states or significant changes in the scene. Each of these keyframes is then individually analyzed by our image captioning model, generating a rich, scene-specific description. This provides a granular level of detail that the broader video summary might omit.

6.4 Enhancing Context with Object Detection

While our captioning models provide high-level summaries of scenes and actions, they may not always capture specific details about every object present in the video. A caption might say, "*A person is working in an office*", but it might not list the specific items on the desk, such as a laptop, a mug, or a phone.

To create a more detailed and robust context for our chatbot, we integrated an **object detection** model. For this task, we chose the **Ultralytics YOLOv11** [21] model, using a version pre-trained on the comprehensive **COCO dataset**. This model is highly efficient and accurate at identifying a wide range of common objects.

The integration process is designed to complement our existing workflow. The same keyframes that are sent to the image captioning model are also processed by YOLOv11. The output from YOLO is a list of all detected objects within those frames. This list is then added to the overall context that is fed to the chatbot.

6.5 Enhancing Context with Speech Recognition

To capture the auditory information within the video, we integrated an automatic speech recognition (ASR) module. For this purpose, we selected PhoWhisper [22], a state-of-the-art model specifically fine-tuned for the Vietnamese language. Based on OpenAI's robust Whisper architecture, PhoWhisper is trained on a large and diverse dataset of Vietnamese accents, ensuring high accuracy for transcription.

The integration process is straightforward: the audio stream is first extracted from the input video and then processed by the PhoWhisper model. The output is a complete text transcript of any spoken dialogue. This transcript is then added to the overall context fed to the chatbot, allowing it to understand and answer questions based on what is said in the video.

6.6 Serving Architecture & Request Flow

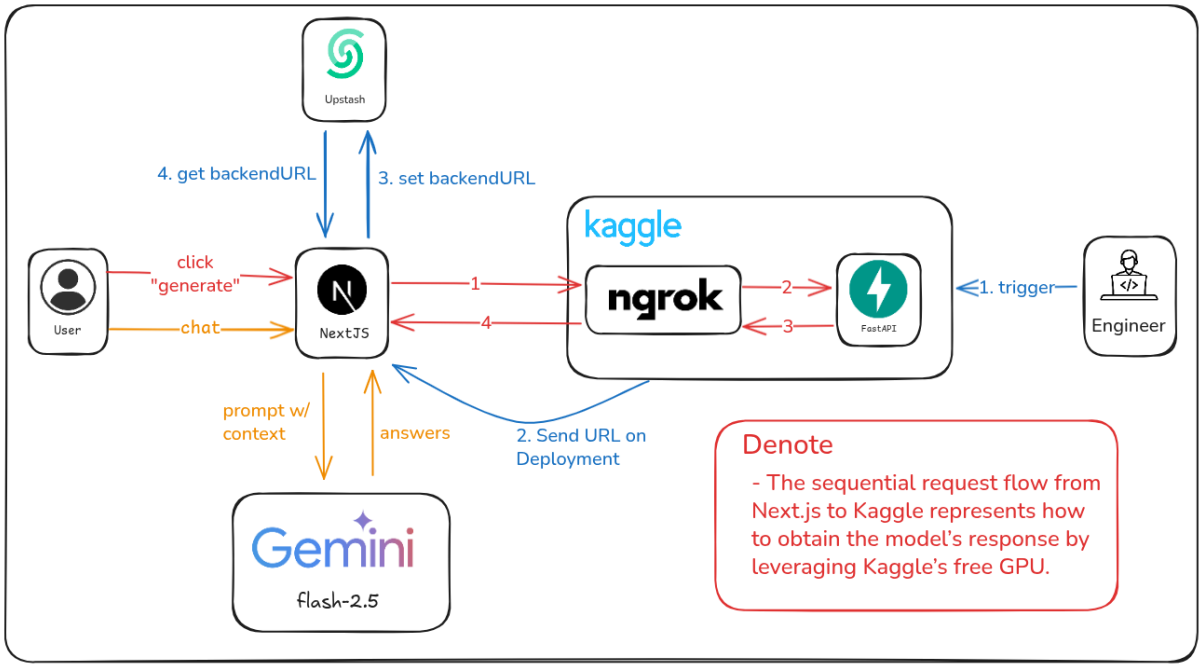

Figure 6: Deployment & Request Flow (Next.js ⇔ Kaggle GPU)

Figure 6 summarizes how the web app serves requests using a lightweight Next.js frontend and a GPU-backed FastAPI service running on Kaggle via ngrok, alongside Gemini 2.5 Flash for conversational reasoning.

Provisioning (blue):

1. An engineer launches the FastAPI service on Kaggle, which is exposed to the Internet through ngrok and yields a public URL.
2. That URL is sent back to the app on deployment.
3. The app stores the URL as `backendURL` in Upstash.
4. At runtime, the Next.js client reads `backendURL` from Upstash so it always targets the current backend.

Inference at runtime (red/orange):

1. When the user clicks **Generate**, Next.js calls the Kaggle `backendURL` through ngrok.
2. Ngrok forwards the request to FastAPI on the GPU.
3. FastAPI returns the result to ngrok.
4. The response is delivered back to Next.js.

In parallel, Next.js sends the prompt plus context to **Gemini 2.5 Flash** and combines the LLM's answer with the GPU model's outputs to reply to the user.

This setup keeps the UI simple, hides the transient Kaggle URL behind Upstash, and lets the system leverage Kaggle's free GPU without operating dedicated servers.

6.7 User Interface

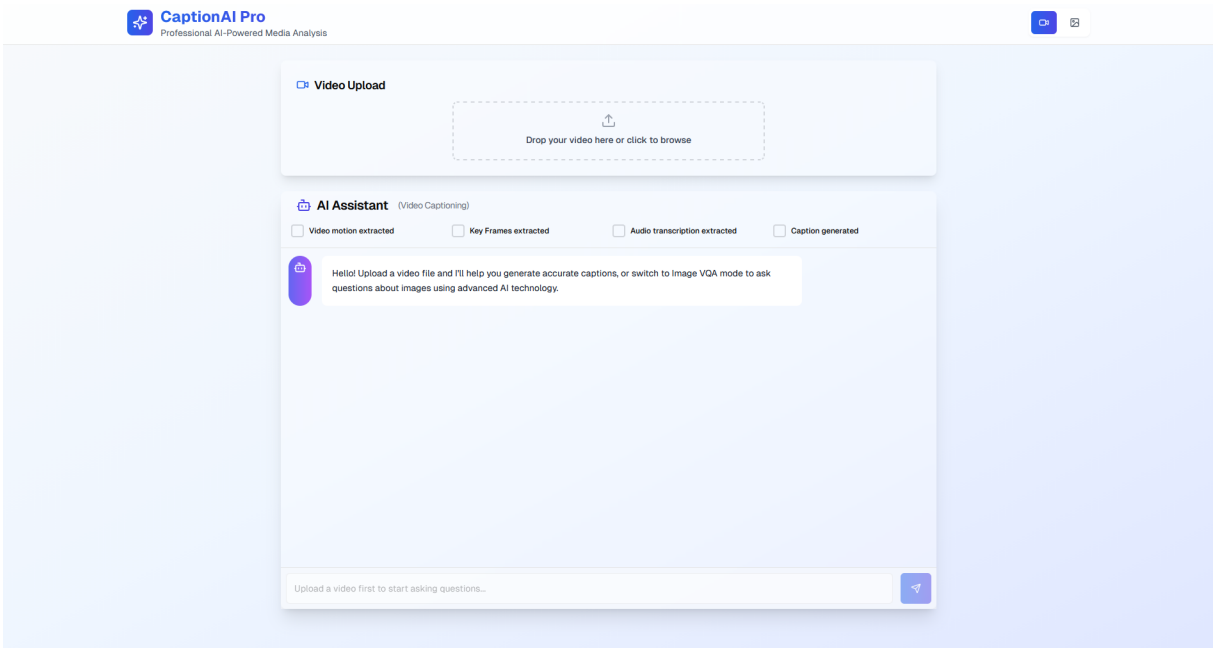

Figure 7: User interface of the our application before uploading video.

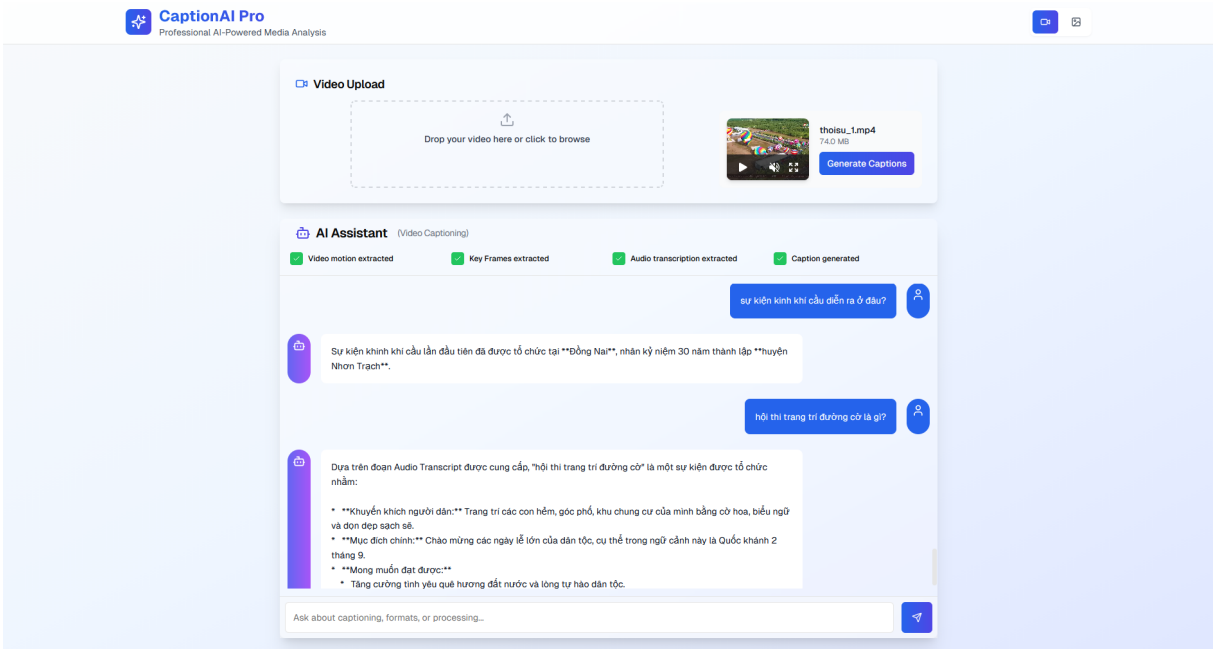

Figure 8: Example of user–chatbot interaction.

7 Limitations

The current system has two primary limitations. The first concerns the replication of the original BTKG results. Our re-implemented baseline did not fully match the published CIDEr-D performance (see Figure 5.2.1), a discrepancy we attribute to differences in computational environments. Despite this gap, the effectiveness of our proposed enhancements is clear, as they consistently outperformed our own baseline. A second limitation is the system's high processing latency, which is almost entirely attributable to the BTKG model. The computational cost of this single video captioning component dominates the end-to-end processing time. Consequently, despite the efficiency of the other modules, the overall system is better suited for offline tasks. For example, generating the video summary for a three-minute clip is a time-intensive task due to BTKG's complexity.

8 Conlusion

In this project, we successfully designed and implemented an intelligent system capable of understanding and answering natural language questions about video content. Our approach is distinguished by its multi-modal context generation, which synthesizes information from four specialized models covering actions, scenes, objects, and speech.

Key achievements include the successful enhancement of the BTKG video captioning model with novel architectural improvements (Feature Fusion and ResiDual connections), which demonstrably improved caption quality over our baseline. The final chatbot application effectively leverages the rich, pre-generated context to provide accurate and detailed answers, proving the efficacy of our integrated architecture.

Although the current implementation faces latency challenges due to the BTKG model's computational demands, our work establishes a powerful proof-of-concept. Future efforts aimed at performance optimization will be crucial for realizing real-time applications. Overall, this project showcases the power of combining multiple AI perspectives to unlock the vast information stored within video data.

Reference

[1] Maosheng Zhong et al. "Bidirectional transformer with knowledge graph for video captioning". In: *Multimedia Tools and Applications* 83 (Dec. 2023), pp. 1–20. DOI: 10.1007/s11042-023-17822-4.

[2] Ashish Vaswani et al. "Attention is all you need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964.

[3] Maosheng Zhong et al. "BiTransformer: augmenting semantic context in video captioning via bidirectional decoder". In: *Mach. Vision Appl.* 33.5 (Sept. 2022). ISSN: 0932-8092. DOI: 10.1007/s00138-022-01329-3. URL: https://doi.org/10.1007/s00138-022-01329-3.

[4] Shufang Xie et al. "ResiDual: Transformer with Dual Residual Connections". In: *ArXiv* abs/2304.14802 (2023). URL: https://api.semanticscholar.org/CorpusID:258418273.

[5] Google. *ViT-Base-Patch16-224-In21k*. https://huggingface.co/google/vit-base-patch16-224-in21k. Accessed: 2025-06-08. 2024.

[6] Alexey Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[7] Colin Raffel et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *Journal of Machine Learning Research* 21.140 (2020), pp. 1–67. URL: http://jmlr.org/papers/v21/20-074.html.

[8] Google. *ViT-Base-Patch16-224*. https://huggingface.co/google/vit-base-patch16-224. Accessed: 2025-04-08. 2024.

[9] Long Phan et al. "ViT5: Pretrained Text-to-Text Transformer for Vietnamese Language Generation". In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Student Research Workshop*. Association for Computational Linguistics, 2022, pp. 136–142. URL: https://aclanthology.org/2022.naacl-srw.18.

[10] Dinh Anh Vu. *Vietnamese COCO 2017 image caption dataset*. 2023. DOI: 10.34740/KAGGLE/DSV/5103962. URL: https://www.kaggle.com/dsv/5103962.

[11] Anh-Cuong Pham et al. "Ktvic: A vietnamese image captioning dataset on the life domain". In: *arXiv preprint arXiv:2401.08100* (2024).

[12] Christian Szegedy et al. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. 2016. arXiv: 1602.07261 [cs.CV].

[13] Joao Carreira and Andrew Zisserman. "Quo vadis, action recognition? a new model and the kinetics dataset". In: *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6299–6308.

[14] Kishore Papineni et al. "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Ed. by Pierre Isabelle, Eugene Charniak, and Dekang Lin. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://aclanthology.org/P02-1040/.

[15] Satanjeev Banerjee and Alon Lavie. "METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments". In: *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. Ann Arbor, Michigan: Association for Computational Linguistics, June 2005, pp. 65–72. URL: https://aclanthology.org/W05-0909/.

[16] Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. "Cider: Consensus-based image description evaluation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 4566–4575.

[17] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013/.

[18] Kaiming He et al. "Mask r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[19] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162/.

[20] Gheorghe Comanici et al. "Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities". In: *arXiv preprint* (2025).

[21] Glenn Jocher and Jing Qiu. *Ultralytics YOLO11*. Version 11.0.0. 2024. URL: https://github.com/ultralytics/ultralytics.

[22] Thanh-Thien Le, Linh The Nguyen, and Dat Quoc Nguyen. "Phowhisper: Automatic speech recognition for vietnamese". In: *arXiv preprint arXiv:2406.02555* (2024).