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

DATA REPRESENTATION

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 9, 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

1.2 Source Code

2 Abstract

3 Model Architecture

This section describes the proposed Bidirectional Transformer with Knowledge Graph (BTKG) model (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.

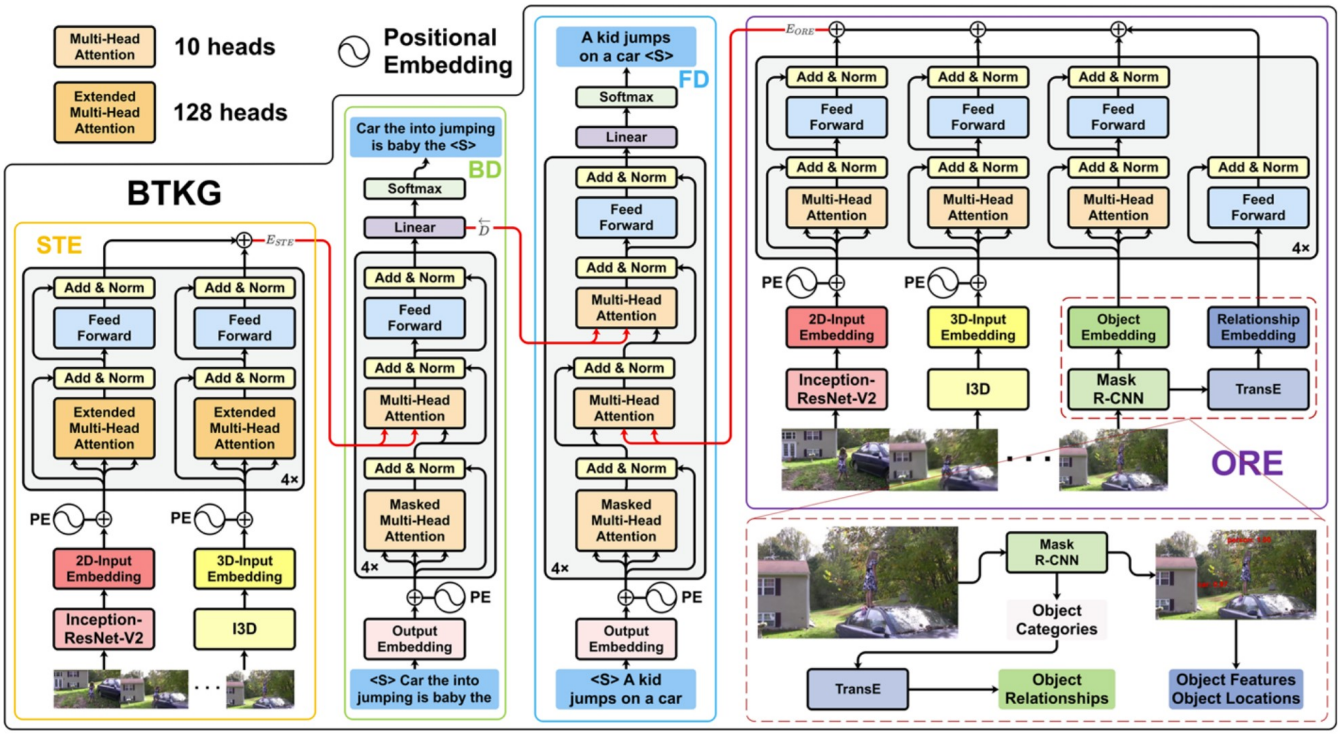

Figure 1: The figure illustrates the proposed BTKG based on the transformer architecture in detail, which consists of Spatio-Temporal Encoder (STE), Objects and Relationships Encoder (ORE), Backward Decoder (BD), and Forward Decoder (FD).

3.1 Basic Module

3.1.1 Transformer Architecture

The model is built upon the standard Transformer [1] 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.

3.1.2 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 [4]: 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.

3.2 Bidirectional Transformer with Knowledge Graph (BTKG)

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.

3.2.1 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.

3.2.2 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).

3.2.3 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.

3.2.4 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 $\overleftarrow{D}$.

3.3 Proposed Enhancements

3.3.1 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 1D convolutional layer (`Conv1D`) 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 along the last dimension. This creates a new tensor of shape `(batch_size, seq_len, n_features * d_model)`, where `n_features` is the number of modalities.
2. **1x1 Convolution**: The concatenated tensor is then passed through a `Conv1D` layer with a kernel size of 1. This convolutional layer serves as a learnable linear projection, mapping the feature space from `n_features * d_model` back down to `d_model`. 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 `Conv1D` 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 `Conv1D` 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.
- **Computational Efficiency**: A 1x1 convolution is a highly efficient operation that does not significantly increase the model's computational cost or complexity but substantially improves its expressive power.

3.3.2 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) [2] at Microsoft. ResiDual is designed to combine the advantages of both Post-LN and Pre-Layer Normalization (Pre-LN) while eliminating their respective drawbacks.

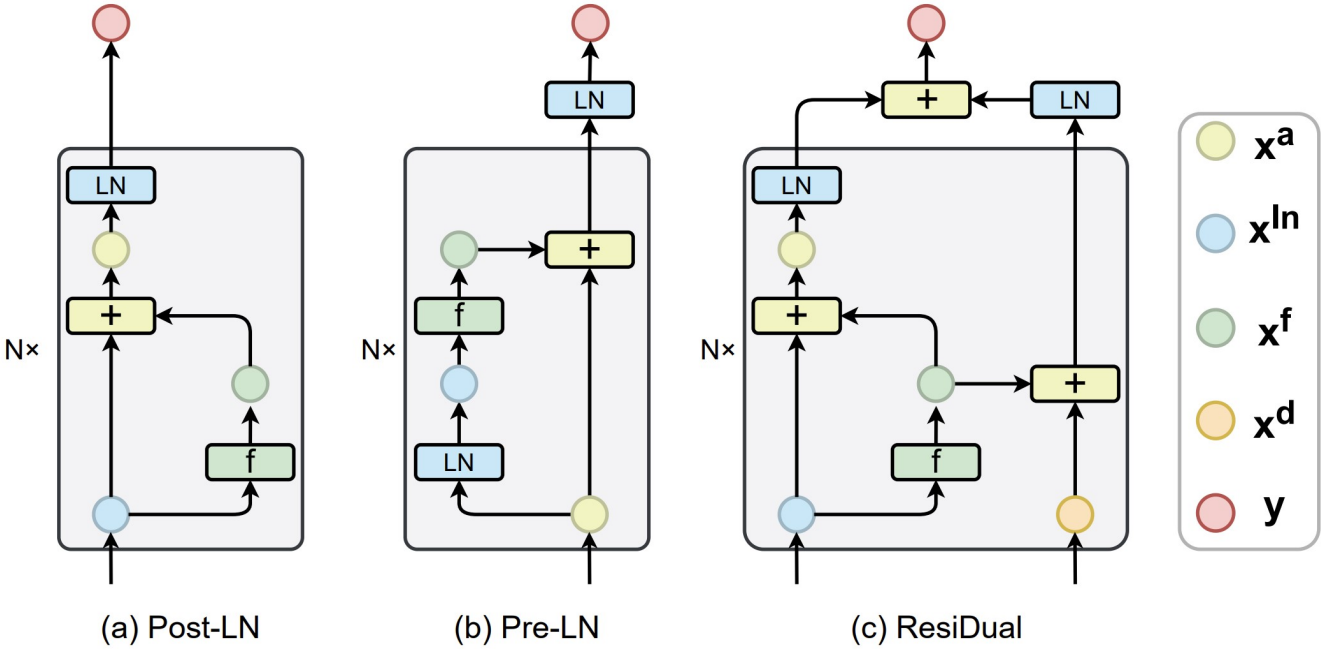

Figure 2: Overview of Post-LN, Pre-LN, and ResiDual. Circles with different colors represent different variables and rectangles represent different operations.

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.

3.4 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.

3.4.1 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.

3.4.2 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.

3.4.3 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 Experiments

4.1 Experimental Settings

4.1.1 Datasets

The **MSVD** dataset (Microsoft Research Video Description) was used as the evaluation benchmark. 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.)

4.1.2 Feature extraction

For MSVD, 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* 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)* 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. For MSVD, 50 frames are uniformly sampled from each video's features, and all feature vectors (image, motion, and category) are projected to a 512-dimensional space.

4.1.3 Evaluation metrics

Performance on MSVD was measured with standard automatic captioning metrics: **BLEU**, **METEOR**, **CIDEr-D**, and **ROUGE-L**. BLEU and METEOR (originally developed for machine

translation) quantify the quality of generated text by n-gram overlap with reference captions. CIDEr-D, explicitly designed for caption evaluation, measures consensus with human annotations. ROUGE-L (longest-common-subsequence based) was also used; it emphasizes recall in overlap between generated and reference sentences. These metrics together provide a comprehensive assessment of caption accuracy, relevance, and fluency.

4.1.4 Parameter settings

We detail the model's settings for MSVD experiments. The object detector (Mask R-CNN) used a confidence threshold of 0.7 and minimum box size of 224x224. A TransE knowledge-graph was built from 613 unique triples, with each relationship encoded by a 300-dimensional GloVe vector. 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. All experiments were run on NVIDIA Tesla P100 GPUs.

4.2 Performance comparison

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.

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

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

4.2.1 Analysis of Results

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.

4.2.2 Conclusion

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 Reference

[1] 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.

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

[3] 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.

[4] 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.