System Architecture and Implementation of a Cross-Modal Transformer Network for Robust Emotion Recognition

Xuan Truong Nguyen

August 2025

1 Linear Projection

Purpose: Align EEG and eye-tracking inputs to a shared 512D space for cross-modal processing.

Input shape:
EEG: $(B \times 74 \times 310)$
Eye: $(B \times 74 \times 33)$ $\quad (B = \text{batch size}, T = \text{time steps, up to } 74)$
Before feeding into the Transformers model, both modalities have to be projected into the same dimensional model space (in our case, 512).

Mathematical Operation:
We denote:
EEG at timestep t: $x_{\text{eeg}}[t] \in R^{310}$
Eye at timestep t: $x_{\text{eye}}[t] \in R^{33}$
Each timestep vector x_t is passed through:

$$z_t = Wx_t + b$$

Where:
W is a learnable weight matrix (shape $[512 \times \text{input\_dim}]$)
b is a learnable bias vector (shape $[512]$)
Both W and b are initialized (typically randomly) and then updated by the optimizer during training.

Gradient computation: During backpropagation, we compute gradients $\frac{\partial \mathcal{L}}{\partial W}$ and $\frac{\partial \mathcal{L}}{\partial b}$ of the loss $\mathcal{L}$ with respect to these parameters.
Update step: The optimizer (Adam) uses those gradients to adjust them. For vanilla gradient descent:

$$W \leftarrow W - \eta \frac{\partial \mathcal{L}}{\partial W}, \quad b \leftarrow b - \eta \frac{\partial \mathcal{L}}{\partial b}$$

where η is the learning rate.

Over many batches/iterations, W and b learn to transform the raw input into useful representations.

Full Linear Projection:

Projected EEG:

$$z_{\text{eeg}}[t] = W_{\text{eeg}} x_{\text{eeg}}[t] + b_{\text{eeg}} \quad \text{where } W_{\text{eeg}} \in R^{512\times 310}$$

Projected Eye:

$$z_{\text{eye}}[t] = W_{\text{eye}} x_{\text{eye}}[t] + b_{\text{eye}} \quad \text{where } W_{\text{eye}} \in R^{512\times 33}$$

After projection:

$$\begin{aligned} \text{EEG} &\to (B, T, 512) \\ \text{Eye} &\to (B, T, 512) \end{aligned}$$

Now both are aligned to the model space — ready for positional encoding.

2 Positional Encoding

2.1 Why Positional Embeddings?

Transformers lack an inherent understanding of sequence order. Unlike RNNs, which process inputs sequentially, Transformers process all timesteps in parallel.

To inject information about the **temporal position** of each input timestep, we introduce **positional encodings**.

This is crucial for time-series data like EEG and eye movement signals, where the position of a signal segment can strongly influence its interpretation.

2.2 Input to Positional Encoding

Before applying positional encoding, both EEG and eye movement features are linearly projected into a common space:

$$\begin{aligned} \text{EEG} &\to (B, T, 512) \\ \text{Eye} &\to (B, T, 512) \end{aligned}$$

Where:
B = batch size
T = sequence length (max = 74 timesteps)
512 = model dimension d_{model}

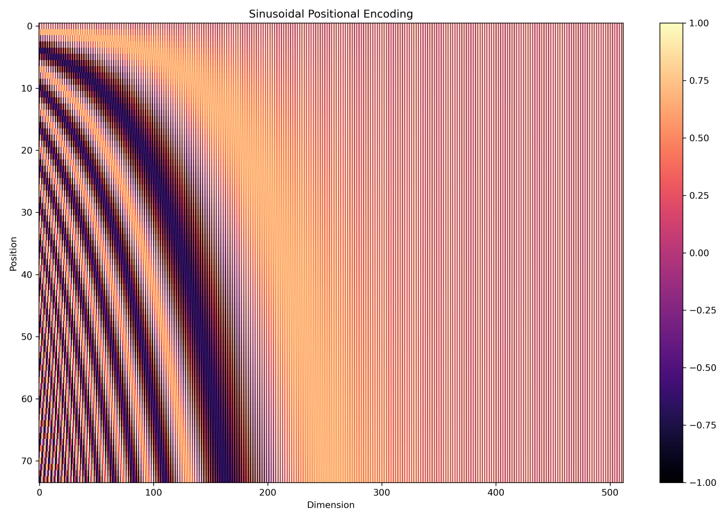

Figure 1: A heatmap of sinusoidal positional encodings across 74 timesteps and 512 dimensions.

2.3 Mathematical Formulation (Embedding Step)

Let:

- $Z \in R^{B \times T \times 512}$: Projected feature matrix (EEG or Eye)
- $P \in R^{T \times 512}$: Positional encoding matrix

Then, we inject position by addition:

$$Z_{\text{pos}} = Z + P$$

Where:

- $P_t \in R^{512}$ is the positional encoding vector for timestep t
- P is broadcasted across the batch dimension

2.4 Intuition and Structure

We use sinusoidal positional encoding as introduced in the original Transformer paper:

Vaswani et al., "Attention is All You Need"

Each position t in the sequence is represented by a fixed vector that alternates between sine and cosine functions with increasing frequencies.

This design:

- Enables the model to learn relative positions and periodic patterns
- Supports generalization to unseen sequence lengths

2.5 Mathematical Formulation (Encoding Calculation)

For position $pos \in [0, T-1]$ and dimension $i \in [0, d_{\text{model}}/2 - 1]$:

$$\text{PE}_{pos,2i} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$
$$\text{PE}_{pos,2i+1} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

Where:

- $d_{\text{model}} = 512$ (in our case)
- $\text{PE}_{pos} \in R^{512}$ is the encoding for timestep pos

Even dimensions get sine functions, odd dimensions get cosine functions.

3 Feature Importance Module (FIM)

3.1 Purpose

The Feature Importance Module is designed to learn modality-specific importance weights that indicate how relevant each modality is for emotion recognition adaptively across different samples, timesteps, and feature dimensions. This helps:

- Emphasize stronger signals (e.g., if EEG is more informative for a sample, weight it more)
- Reduce reliance on noisy or less relevant modalities

3.2 Architectural Role

Assume that after linear projection and positional encoding, we obtain:

$$Z_{\text{eeg}} \in R^{B\times T\times d}$$
$$Z_{\text{eye}} \in R^{B\times T\times d}$$

These represent the EEG and eye-tracking features, respectively, where:

- B: Batch size
- T: Sequence length (timesteps)
- d: Feature dimension after projection

These modality-specific features are processed independently at first. Before fusion or cross-modal attention, we apply the Feature Importance Module to reweight each feature tensor.

3.3 Mathematical Formulation

1. Modality-Specific Importance Estimation

We use a small feedforward network to compute a scalar importance score for each modality at each timestep.

Let:

- $W_f \in R^{d\times 1}$, $b_f \in R$

Then, for each timestep $t \in [1, T]$, compute:

$$\alpha_{\text{eeg}}[t] = \sigma(Z_{\text{eeg}}[t]W_f + b_f) \in R$$
$$\alpha_{\text{eye}}[t] = \sigma(Z_{\text{eye}}[t]W_f + b_f) \in R$$

where $\sigma(\cdot)$ denotes the sigmoid activation function to squash values to the range $[0, 1]$.

2. Reweighting the Modalities

We then reweight each modality's features using the learned importance scores:

$$\tilde{Z}_{\text{eeg}}[t] = \alpha_{\text{eeg}}[t] \cdot Z_{\text{eeg}}[t]$$
$$\tilde{Z}_{\text{eye}}[t] = \alpha_{\text{eye}}[t] \cdot Z_{\text{eye}}[t]$$

These rescaled embeddings $\tilde{Z}_{\text{eeg}}$ and $\tilde{Z}_{\text{eye}}$ are passed to the subsequent fusion or attention modules in the model.

4 Cross-Modal Attention Mechanism

To capture the dependencies between EEG and eye movement signals, we employ a bidirectional cross-modal attention mechanism. Specifically, we allow EEG features to attend to eye movement features and vice versa. This helps the model learn interactions between modalities at each timestep.

4.1 Multi-Head Attention Setup

Let $Z^{\text{EEG}}, Z^{\text{Eye}} \in R^{B\times T\times D}$ denote the EEG and Eye input features, where B is the batch size, T is the sequence length, and D is the feature dimension. We define h as the number of attention heads, and $d_h = D/h$ as the dimension per head.

These matrices $\bar{A} \in R^{B\times T\times T}$ represent how timesteps in one modality attend to another and can be visualized as attention heatmaps.

Linear Projections

We first project the input features into queries, keys, and values:

$$\begin{aligned} Q^{\text{EEG}} &= Z^{\text{EEG}} W_Q^{\text{EEG}}, \quad & K^{\text{EEG}} &= Z^{\text{EEG}} W_K^{\text{EEG}}, \quad & V^{\text{EEG}} &= Z^{\text{EEG}} W_V^{\text{EEG}} \\ Q^{\text{Eye}} &= Z^{\text{Eye}} W_Q^{\text{Eye}}, \quad & K^{\text{Eye}} &= Z^{\text{Eye}} W_K^{\text{Eye}}, \quad & V^{\text{Eye}} &= Z^{\text{Eye}} W_V^{\text{Eye}} \end{aligned}$$

where $W_Q, W_K, W_V \in R^{D \times D}$ are learnable projection matrices.

Split into Heads

We reshape these projections into h attention heads:

$$Q \in R^{B \times h \times T \times d_h}, \quad K \in R^{B \times h \times T \times d_h}, \quad V \in R^{B \times h \times T \times d_h}$$

4.2 Attention Computation

We compute scaled dot-product attention between the two modalities in both directions.

EEG → Eye Attention

$$A^{\text{EEG}\to\text{Eye}} = \text{softmax}\left(\frac{Q^{\text{EEG}} (K^{\text{Eye}})^\top}{\sqrt{d_h}} + M^{\text{Eye}} \right) \in R^{B \times h \times T \times T}$$

Here, M^{Eye} is the eye attention mask that sets positions corresponding to padded timesteps to $-\infty$.

$$\text{EEG attended} = A^{\text{EEG}\to\text{Eye}} \cdot V^{\text{Eye}} \in R^{B \times h \times T \times d_h}$$

Eye → EEG Attention

$$A^{\text{Eye}\to\text{EEG}} = \text{softmax}\left(\frac{Q^{\text{Eye}} (K^{\text{EEG}})^\top}{\sqrt{d_h}} + M^{\text{EEG}} \right) \in R^{B \times h \times T \times T}$$

Here, M^{EEG} is the eye attention mask that sets positions corresponding to padded timesteps to $-\infty$.

$$\text{Eye attended} = A^{\text{Eye}\to\text{EEG}} \cdot V^{\text{EEG}} \in R^{B \times h \times T \times d_h}$$

4.3 Output Projection and Residual Fusion

We concatenate the heads and project back to the original dimension:

$$\begin{aligned} \hat{Z}^{\text{EEG}} &= Z^{\text{EEG}} + \text{Proj}\left(\text{ConcatHeads}(\text{EEG attended})\right) \\ \hat{Z}^{\text{Eye}} &= Z^{\text{Eye}} + \text{Proj}\left(\text{ConcatHeads}(\text{Eye attended})\right) \end{aligned}$$

Here, $\text{Proj}(\cdot)$ denotes a linear transformation back to $R^{B \times T \times D}$, and residual connections are used to preserve the original modality features.

4.4 Interpretability

For interpretability, we average the attention weights across heads:

$$\bar{A}^{\text{EEG}\to\text{Eye}} = \frac{1}{h}\sum_{i=1}^{h} A_i^{\text{EEG}\to\text{Eye}}, \quad \bar{A}^{\text{Eye}\to\text{EEG}} = \frac{1}{h}\sum_{i=1}^{h} A_i^{\text{Eye}\to\text{EEG}}$$

These matrices $\bar{A} \in R^{B\times T\times T}$ represent how timesteps in one modality attend to another and can be visualized as attention heatmaps.

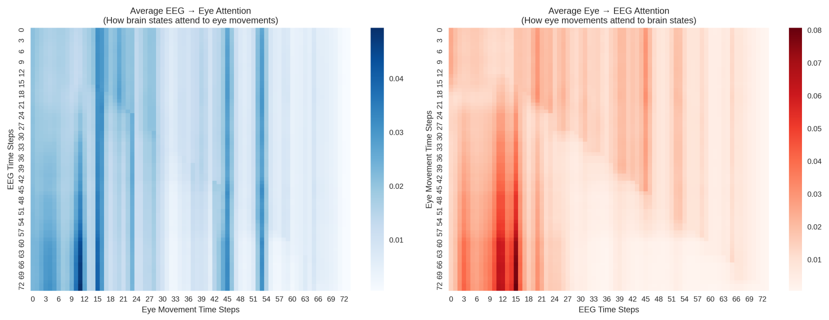

Figure 2: Cross-modal attention heatmaps. Left: EEG attending to Eye. Right: Eye attending to EEG.

5 Self-Attention Transformer Encoder

After cross-modal interaction, we apply a Transformer Encoder to each modality to enable intra-modal contextual modeling across time steps. Each encoder layer consists of a Multi-Head Self-Attention block and a Position-wise Feedforward Network, each wrapped in residual connections and layer normalization.

Multi-Head Self-Attention

Let the input be $Z \in R^{B\times T\times D}$, where:

- B: batch size
- T: number of timesteps
- D: hidden dimension

For each attention head $i \in \{1, \ldots, h\}$, we compute:

$$Q_i = ZW_i^Q \in R^{B\times T\times d_h}$$
$$K_i = ZW_i^K \in R^{B\times T\times d_h}$$
$$V_i = ZW_i^V \in R^{B\times T\times d_h}$$

where:

- $d_h = \frac{D}{h}$: dimension per head
- $W_i^Q, W_i^K, W_i^V \in R^{D \times d_h}$: learnable projections

We then compute scaled dot-product attention:

$$\text{Attention}_i = \text{softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d_h}} + M\right) V_i$$

Here, $M \in R^{B \times T \times T}$ is an optional attention mask, with values set to $-\infty$ for padded positions and 0 elsewhere.

The multi-head attention output is computed by concatenating all heads and applying a final linear projection:

$$\text{MultiHead}(Z) = \text{Concat}(\text{Attention}_1, \ldots, \text{Attention}_h) W^O \in R^{B \times T \times D}$$

Position-wise Feedforward Network

A two-layer feedforward network is applied independently to each timestep:

$$\text{FFN}(x) = \text{Dropout}(W_2(\text{GELU}(W_1 x)) + b_2) \in R^{B \times T \times D}$$

where:

- $W_1 \in R^{D \times D_{ff}}$, $W_2 \in R^{D_{ff} \times D}$
- D_{ff}: inner hidden size (e.g., 1024)

Final Encoder Output

Residual connections and layer normalization are applied as follows:

$$\begin{aligned} Z' &= \text{LayerNorm}(Z + \text{Dropout}(\text{MultiHead}(Z))) \\ Z_{\text{out}} &= \text{LayerNorm}(Z' + \text{FFN}(Z')) \end{aligned}$$

The resulting $Z_{\text{out}} \in R^{B \times T \times D}$ is forwarded to Global Pooling and Fusion stage.

6 Global Pooling and Fusion

6.1 Global Pooling

After modality-specific Transformer encoders, we extract a fixed-size representation for each modality by applying global average pooling across the temporal dimension:

$$\begin{aligned} g_{\text{EEG}} &= \text{MeanPooling}(Z_{\text{out}}^{\text{EEG}}) \in R^{B \times D} \\ g_{\text{Eye}} &= \text{MeanPooling}(Z_{\text{out}}^{\text{Eye}}) \in R^{B \times D} \end{aligned}$$

Here, $Z_{\text{out}}^{\text{EEG}}, Z_{\text{out}}^{\text{Eye}} \in R^{B\times T\times D}$ are the encoder outputs, and the pooling operation averages over the time dimension T to produce modality-specific embeddings.

6.2 Feature Fusion

The final multimodal representation is obtained by concatenating the two pooled features:

$$g_{\text{fused}} = [g_{\text{EEG}}; g_{\text{Eye}}] \in R^{B\times 2D}$$

This fused vector integrates information from both modalities and serves as input to the Domain Adaptation layer.

7 Domain Adaptation Layer

In our multimodal Transformer model, the goal is to generalize emotion recognition across different subjects (i.e., domains). Since EEG and Eye Movement features vary significantly between individuals, the domain adaptation layer enables the model to learn subject-invariant representations, improving generalization to unseen users.

7.1 Architecture Overview

The domain adaptation component comprises three key parts:

1. **Feature Extractor** — the main Transformer model including EEG and Eye modality encoders, cross-modal attention, and fusion.
2. **Emotion Classifier** — predicts the emotion label from the fused representation.
3. **Domain Classifier** — predicts the subject identity (i.e., domain) from the same fused representation. This component is trained adversarially via a Gradient Reversal Layer (GRL).

7.2 Mathematical Formulation

Let:

- $g \in R^{B\times 2D}$: fused representation after global pooling and concatenation.
- $f_{\text{cls}}(g)$: emotion classifier output.
- $f_{\text{dom}}(g)$: domain classifier output.
- y: emotion label.
- d: domain label (subject ID).

1. Emotion Classification Loss

$$\mathcal{L}_{\text{cls}} = \frac{1}{B} \sum_{i=1}^{B} \text{CrossEntropy}(f_{\text{cls}}(g_i), y_i)$$

2. Domain Classification with Gradient Reversal Before feeding into the domain classifier, g passes through a GRL:

$$g^{\text{rev}} = \text{GRL}(g)$$

During backpropagation, GRL multiplies gradients by $-\alpha$, where α increases during training. The domain classification loss is:

$$\mathcal{L}_{\text{dom}} = \frac{1}{B} \sum_{i=1}^{B} \text{CrossEntropy}(f_{\text{dom}}(g_i^{\text{rev}}), d_i)$$

3. Total Loss (Joint Objective)

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{cls}} + \lambda \cdot \mathcal{L}_{\text{dom}}$$

Here, λ is a fixed hyperparameter (e.g., 0.1) controlling the relative weight of domain adaptation in the total loss.

7.3 Gradient Reversal Layer (GRL)

The GRL is a special operation with the following behavior:

- **Forward pass:** Identity function — $\text{GRL}(g) = g$
- **Backward pass:** Gradient is reversed and scaled:

$$\frac{\partial \mathcal{L}}{\partial g} \to -\alpha \cdot \frac{\partial \mathcal{L}}{\partial g}$$

This adversarial mechanism causes the feature extractor to:

- Minimize $\mathcal{L}_{\text{cls}}$: learn features useful for emotion classification.
- Maximize $\mathcal{L}_{\text{dom}}$: learn features that confuse the domain classifier, encouraging domain-invariant representations.

7.4 Dynamic α Scheduling

To gradually increase domain-adversarial learning, we schedule the GRL strength α using a sigmoid ramp-up:

$$\alpha(e) = \frac{2}{1 + \exp(-10 \cdot \frac{e}{E})} - 1$$

Where:

- e: current epoch
- E: total number of training epochs

Behavior:

- **Early epochs:** $\alpha \approx 0$ — GRL has little effect, allowing the model to focus on learning emotion-discriminative features.
- **Later epochs:** $\alpha \to 1$ — GRL becomes stronger, encouraging domain-invariant representation learning.

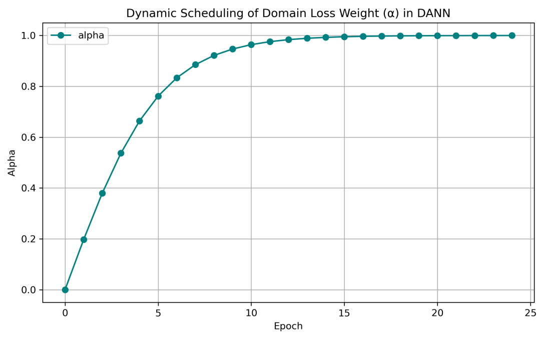

Figure 3: Dynamic α scheduling during training. As training progresses, α increases smoothly to strengthen the adversarial signal from the GRL.

7.5 Subject-Specific Normalization

This handles inter-subject distribution shifts after adversarial training.

Given a feature vector $x \in R^d$ for subject i, the normalized output is:

$$\text{SubjectNorm}^{(i)}(x) = \gamma^{(i)} \cdot \frac{x - \mu}{\sigma} + \beta^{(i)}$$

Where:

$$\mu = \frac{1}{d} \sum_{j=1}^{d} x_j$$

$$\sigma = \sqrt{\frac{1}{d} \sum_{j=1}^{d} (x_j - \mu)^2 + \epsilon}$$

- d: dimensionality of the input (e.g., 512)
- $\gamma^{(i)}, \beta^{(i)} \in R^d$: learnable parameters for each subject i
- ϵ: small constant for numerical stability

8 Classification Head

The **Classification Head** maps the fused multimodal representation $g \in R^{B\times 2D}$ to the emotion label space. This is implemented as a multilayer perceptron (MLP) with GELU activations and dropout regularization.

8.1 Architecture Overview

The classification head consists of three fully connected layers:

$$\begin{aligned}
h_1 &= \text{GELU}(W_1 g + b_1) && \in R^{B\times 256} \\
h_1' &= \text{Dropout}(h_1) \\
h_2 &= \text{GELU}(W_2 h_1' + b_2) && \in R^{B\times 128} \\
h_2' &= \text{Dropout}(h_2) \\
\hat{y} &= \text{Softmax}(W_3 h_2' + b_3) && \in R^{B\times C}
\end{aligned}$$

where:

- $g \in R^{B\times 2D}$ is the fused representation (EEG + Eye).
- B is the batch size.
- D is the feature dimension of each modality.
- C is the number of emotion classes.
- W_i, b_i are learnable parameters of the linear layers.
- GELU is the Gaussian Error Linear Unit activation.
- Dropout is applied after each nonlinearity to prevent overfitting.

8.2 PyTorch Implementation

```
self.classifier = nn.Sequential(
    nn.Linear(d_model*2, 256),
    nn.GELU(),
    nn.Dropout(dropout),
    nn.Linear(256, 128),
    nn.GELU(),
    nn.Dropout(dropout),
    nn.Linear(128, n_classes)
)
```

8.3 Loss Function

The predicted class probabilities $\hat{y}$ are compared to the ground truth emotion labels y using the standard cross-entropy loss:

$$\mathcal{L}_{\text{cls}} = \frac{1}{B} \sum_{i=1}^{B} \text{CrossEntropy}(\hat{y}_i, y_i)$$

This loss is combined with the domain loss $\mathcal{L}_{\text{dom}}$ during joint training.