#### ***CV Project Report***

## **Code Segments and Modifications**

### **1. XRGEN Model**

#### **Original Functionality**

The XRGEN model integrates multi-modal input (visual features, textual embeddings, and topic embeddings) for advanced medical image understanding. It was initially functional but lacked clarity in handling various components and modes.

#### **Key Modifications**

* **Device Handling**: Added a device argument to dynamically manage tensor placement on GPU or CPU.
* **Topic Embedding Management**: Refactored topic embedding setup into reusable functions setup\_topics and merge\_topic\_embeddings.
* **Improved Forward Pass Logic**: Modularized training and sampling logic into separate code blocks with helper functions.
* **Logging**: Introduced logging for key operations like embedding setups, pre-trained weight loading, and device configuration.

#### **Impact**

* Enhanced code clarity and reusability.
* Improved debugging through added logging.
* Better adaptability for different hardware setups.

### **2. Contrastive Loss Computation**

#### **Original Functionality**

The original code computed contrastive losses directly within the forward pass, which led to mixed concerns and a less modular implementation.

#### **Key Modifications**

* Separated contrastive loss computation into a dedicated compute\_logits method.
* Added safeguards to ensure the logit scale remains within a valid range using torch.clamp.

#### **Impact**

* Simplified forward pass logic.
* Ensured robust scaling of contrastive learning outputs.

### **3. Text Encoder**

#### **Original Functionality**

Textual embeddings were computed directly during the forward pass, without abstraction or pre-processing utilities.

#### **Key Modifications**

* Refactored the text encoder into a dedicated class with methods for embedding tokenization and processing.
* Added pre-tokenized inputs for efficiency during training.

#### **Impact**

* Improved modularity for pre-processing textual inputs.
* Reduced repetitive computations during training and inference.

### **4. Visual Feature Extraction**

#### **Original Functionality**

The visual extractor used a monolithic method to handle feature computation.

#### **Key Modifications**

* Introduced modular helper functions for:
  + Feature normalization.
  + Batch-wise feature extraction.
* Added support for optional fine-tuning of pre-trained weights.

#### **Impact**

* Enhanced flexibility for different image datasets.
* Better memory management during batch processing.

### **5. Sequence Preparation for Encoder-Decoder**

#### **Original Functionality**

Sequence preparation logic (e.g., cropping and masking) was embedded directly in the forward pass, making the code harder to extend.

#### **Key Modifications**

* Refactored sequence preparation into a helper function prepare\_seq.
* Added options for configurable sequence lengths and masking strategies.

#### **Impact**

* Improved code readability and extensibility.
* Enabled customization for sequence handling in diverse datasets.

### **6. Streamlit Frontend Integration**

#### **Original Functionality**

A basic Streamlit frontend was planned for user interaction but lacked a modular backend interface.

#### **Key Modifications**

* Developed a modular backend API for model interaction.
* Added real-time logging of predictions and user interactions.
* Included error handling for invalid inputs.

#### **Impact**

* Streamlined user experience with intuitive input handling.
* Enhanced debugging with real-time logging.

### **7. Dataset Preprocessing and BLEU Score Calculation**

#### **Original Functionality**

Dataset preprocessing and BLEU score calculation were handled directly in the main training script, reducing clarity and flexibility.

#### **Key Modifications**

* Abstracted preprocessing into standalone modules:
  + load\_dataset: For loading and splitting datasets.
  + transform\_data: For data augmentation and normalization.
* Added a dedicated compute\_bleu function for evaluating model outputs.

#### **Impact**

* Improved maintainability and reusability of data processing logic.
* Simplified integration of BLEU score evaluation in different workflows.

### **8. General Logging Enhancements**

#### **Original Functionality**

Logging was minimal and inconsistent across components, making debugging challenging.

#### **Key Modifications**

* Standardized logging across all modules:
  + Device configurations.
  + Data loading status.
  + Training progress and evaluation metrics.
* Added verbosity levels (e.g., INFO, DEBUG, ERROR).

#### **Impact**

* Improved traceability and debugging.
* Enhanced visibility into model training and evaluation processes.

### **9. Multi-Modal Weighted Fusion**

#### **Original Functionality**

The fusion logic for multi-modal data (e.g., X-ray and MRI images) was hard-coded and lacked flexibility for dynamic weight adjustments.

#### **Key Modifications**

* Introduced a weighted fusion function that adjusts modality weights based on resolution or accuracy metrics.
* Added support for dynamic re-weighting during training based on feedback loops.

#### **Impact**

* Enhanced fusion accuracy by leveraging dynamic weight adjustments.
* Improved scalability for integrating additional modalities.

## **Conclusion**

The comprehensive modifications across all provided codes have resulted in a more robust, modular, and efficient framework. Key highlights include improved device compatibility, modularized topic embedding handling, dynamic fusion strategies, and an intuitive frontend for user interaction. These enhancements collectively contribute to better maintainability, scalability, and user experience, ensuring the project's success in both experimental and real-world deployments.