MOSAIC-(PS 2): Detailed Report of the Solution

The problem statement revolves around the creation of a multimodal AI model. The task is for a given dataset containing multiple choice geometrical and mathematical problems in the form of images along with problem description and these problems are first to be analysed using techniques like computer vision and natural language processing to interpret their semantic and wholesome context. Thereafter they are to be fed to a deep learning/reasoning model to finally reason and apply logic and predict an answer out of the four available options.

Inspiration from existing models and frameworks

Inter-GPS(Interpretable Geometry Problem Solver)

- It is a novel geometry problem solving approach.
- Inter-GPS parses the problem text and diagram into formal language automatically via rule-based text parsing and neural object detecting, respectively.
- Unlike implicit learning in existing methods, Inter-GPS incorporates **theorem knowledge** as conditional rules and performs symbolic reasoning step by step

This is the canonical approach to solve the geometry problems in the geometry 3k dataset.

Why can it not be incorporated currently in the model?

As per the rules of the PS, we are only to use the dataset provided to us for all the training,analysing,preprocessing tasks. Since the highlight of Inter-GPS is step by step symbolic reasoning based on rule based conditionals and theorems along with implicit learning, implementing this would naturally require a **database of geometry problems (following a heuristic approach)** or a **RAG based knowledge graph creation by which the reasoning model retrieves such information and dynamically includes it in its chain of thought to simulate a step by step symbolic reasoning process**.(This approach mimics the way humans actually solve

math problems-understanding problem context-step by step symbolic and theorem based reasoning leading to the final answer.

State of the Art Models based on this approach:

GAPS Architecture

1. Problem-Type Classifier:
    - GAPS includes a classifier to identify the type of geometry problem (e.g., calculation or proving) and adapt its solution approach accordingly- eliminated for our purpose, since we have to simply calculate the answer and predict the right option
2. Image Processing:
    - If the input involves diagrams or images, GAPS uses image processing techniques to extract geometric information such as points, lines, and shapes.
3. Text Processing:
    - For text-based inputs, GAPS employs natural language processing (NLP) to parse and understand the geometric problem description.
4. Geometry Representation:
    - Converts both image and text inputs into a unified geometric representation that captures key elements and relationships.
5. Problem Encoding:
    - Encodes the geometric representation into a format suitable for processing by the model, often using vector representations.
6. Problem Solving:
    - Applies geometric reasoning and rules to solve the encoded problem, generating a step-by-step solution.
7. Solution output:
    - Formats the solution into a human-readable form, which may include diagrams, equations, or textual explanations- again not needed for this task

Pertaining to the resource limitations(GPU limits) and dataset constraint the approach applied is inspired by the multiple layers of the GAPS framework implemented via fine-tuning existing models and test and validation based training of this entire custom model using back propagation.

Model Architecture and Approach:

The Model architecture implemented is a comprehensive multilayered model design in an attempt to simulate the architecture of GAPS-

First comes the image processing layer. The task of analysing the geometrical images and their labels and co-relate them with the problem description is the foundation of decoding the semantics and context of the question to begin with. Images in this dataset can be interpreted as majorly consisting of two parts:-

- The geometric figure(arc, lines,polygons, circles, shaded area etc.)
- The corresponding text labellings(named angles, vertices etc.)

Thus the computer vision task has been subdivided into two subtasks:-

1. The first task is to understand the geometric skeleton(foundational diagram-like the polygons, parallel lines intersected by a transversal etc.). This is an apt CNN task for detecting shapes and figures. Thus for this purpose, **EfficientB4Net** has been fine-tuned(using ***Lora***).

   EfficientNet-B4 is a variant within the EfficientNet family of convolutional neural networks, optimized for image classification tasks. It uses *compound scaling* to uniformly adjust network depth, width, and resolution, achieving high accuracy with computational efficiency. EfficientNet-B4 has 19 million parameters.

   A custom EfficientNetB4 Adapter has been developed which performs the following operations:-

- Defines a wrapper class (`EfficientNetAdapter`) that extracts convolutional features from a pre-trained EfficientNet-B4 and adds adaptive pooling.

- Uses `peft` to apply LoRA to specific layers (`fc1`, `fc2`) in the network's feature extractor, keeping original weights frozen while training low-rank adapters.
- Configures LoRA with rank=16, alpha=32, and 5% dropout for parameter-efficient adaptation to new tasks.
- The `fc1` and `fc2` layers mentioned in the LoRA configuration are likely intended to represent fully connected layers (dense layers) in the EfficientNet-B4 architecture
- Fully connected layers typically appear in the final classification head of a neural network, where features extracted by convolutional layers are mapped to class probabilities or task-specific outputs

2. The second task is to extract the text embedded in these images, the labellings of angles and vertices and for this purpose easyocr, an open-source **Optical Character Recognition (OCR)** library built on PyTorch, designed to extract text from images and scanned documents.

   This is achieved by performing the following operations:

- Initializing an **EasyOCR** Reader to perform OCR on English text, using GPU if `DEVICE` is set to `'cuda'`.
- Processing an image (`item['images']`) using the `readtext` method, extracting the detected text from the results, and joining them into a single string (`ocr_text`).

Preprocessing techniques involved:

- Image Preprocessing: Converts RGBA images to RGB, resizes to 380x380, and converts to PyTorch tensors.
- Label Encoding: Converts ground-truth answers (A/B/C/D) to numerical indices (0/1/2/3).

- BERT Tokenization: Processes combined text with BERT tokenizer (max 512 tokens, padded) for NLP input.
- Option Handling: Attempts numeric conversion of choice values for hybrid loss calculation, skips non-numeric cases(the string based options are sent via raw_choices for hybrid loss calculation purpose-explained later in this report)

3. The next step is to process the text, that is the problem description. This is a sequence to sequence natural language processing task and not only the text is to be processed, but all the information in the text captured as embeddings must be correlated with the visual features as captured by the visual embeddings along with the embeddings of the label text detected via OCR based detection. For this purpose, we have:

- First the problem description is encoded to an embedding space using a text encoder in the form of **BERT**.
- Bert-base-uncased is fine tuned via lora, with r=16, r_alpha=32 and the layers **query and value (with reference to the attention mechanism of the encoder)** are specifically trained to learn the text even better with training
- The next step is to fuse these multiple embeddings to understand the wholesome context-this is achieved via a fusion layer

3. Fusion layer or multihead attention layer:

- The `FusionLayer` merges visual and textual information into a unified representation for multimodal tasks. It first projects image features (e.g., from a CNN) and text embeddings (e.g., OCR text or problem descriptions) into a shared 512-dimensional space using separate linear layers. These aligned features are then processed by multi-head attention, where the image features act as "queries" to dynamically focus on the most relevant parts of the text ("keys" and "values"). This attention mechanism creates a context-aware fusion of visual

and textual cues. Finally, the fused output is projected into a richer 2048-dimensional space, enhancing its expressiveness for downstream tasks like answering questions that require joint understanding of images and text.

4. ***Reasoning model employed-lightweight LLM (phi-1.5)***:

   Phi-1.5 is a 1.3 billion parameter Transformer-based language model developed by Microsoft, designed for tasks requiring natural language understanding, reasoning, and code generation. It builds on its predecessor, Phi-1, by leveraging high-quality synthetic "textbook-like" data instead of traditional web-crawled data, enhancing safety and reducing biases. Despite its relatively small size, Phi-1.5 *achieves performance comparable to models five times larger on benchmarks like common sense reasoning and logical tasks*. Its efficiency, versatility, and open-source nature make it ideal for this task. Specifically considering the multimodal nature and complexity of the problem- the major issue is optimization of resources(at most the use of t4 gpu in colab)- thus a lightweight model with high accuracy was the most suitable choice. *Also it has majorly been trained on textbook data (science,maths etc)-making it ideal for elementary geometry problems*

   The model is finetuned via lora, which trains only specific layers while freezing the rest of the model.

Layers Being Trained:

1. `q_proj`: The *query projection layer* in the transformer's attention mechanism.
2. `k_proj`: The *key projection layer* in the attention mechanism.

These layers are part of the self-attention blocks and are modified via low-rank matrices during fine-tuning. All other layers (e.g., `v_proj`, MLP layers, embeddings) remain frozen to retain pretrained knowledge.

Why These Layers?

- Attention Focus: The `q_proj` and `k_proj` layers determine how the model attends to different parts of the input. Adapting them allows task-specific focus without overfitting[1][4][6].
- Parameter Efficiency: Training just these layers reduces trainable parameters by ~99% compared to full fine-tuning, enabling GPU-friendly training (e.g., on T4 GPUs)

5. The use of T5-decoder:

   The combined embeddings as obtained from the multihead layer needs to be decoded to text in order to be fed as a text based prompt to the LLM based upon which it can make the prediction. **T5-small decoder** has been utilized for this purpose. During the training, it is trained via **teacher forcing.**

   The decoder receives the *ground-truth target sequence* shifted right (prepended with a start token) as `decoder_input_ids`, while the original sequence (with an EOS token) serves as `labels`. This forces the model to predict the next token based on the true prior tokens, not its own outputs, improving gradient stability. Cross-entropy loss is computed between decoder predictions and actual labels. This method enables efficient training for text-to-text tasks by aligning inputs with expected outputs.

6. **Full stack training of the custom multimodal model via backpropagation**:

- Data Preparation and Feature Extraction
  The training process starts by preparing the data for both visual and

textual modalities. Images are preprocessed to ensure they are in RGB format (grayscale images are converted by repeating channels). Text inputs, including problem descriptions and options, are tokenized into input IDs and attention masks using a text encoder (e.g., BERT). These preprocessed inputs are then passed through separate encoders: an image encoder extracts features from the images, which are flattened into a fixed-dimensional vector, while the text encoder generates embeddings for the textual inputs. The extracted features are used as the foundation for multimodal reasoning.

- Fusion of Visual and Textual Features
  The extracted image and text features are projected into a shared 512-dimensional space using linear layers. These aligned features are then fused using a fusion layer, which employs multi-head attention to combine visual and textual contexts dynamically. The fused representation is expanded to include a sequence dimension (`unsqueeze(1)`) and serves as input to a decoder (e.g., Phi-1.5). This fusion ensures that the model can reason jointly about both modalities, enabling it to understand complex relationships between images and text.
- Decoder Input and Forward Pass
  The decoder is set up using teacher forcing, where ground-truth labels (answer indices) are prepended with a start token (`decoder_input_ids`). The fused multimodal representation is passed to the decoder, which predicts logits for the possible answer options (A/B/C/D). The forward pass computes these logits by attending to both visual and textual contexts, enabling the model to generate predictions based on the combined information. This step is crucial for tasks like answering questions that require reasoning across both modalities.
- Loss Calculation and Hybrid Loss Design
  The training uses two types of losses:

- Cross-Entropy Loss (CE): Measures how well the model predicts the correct answer index out of the 4 options.
- Hybrid Loss: If numeric values for options are available, Mean Squared Error (MSE) compares predicted values with ground truth. If only text options are available, Cosine Embedding Loss calculates similarity between predicted embeddings and correct embeddings generated by the text encoder. The final loss is a weighted combination of CE loss (40%) and hybrid loss (60%), balancing classification accuracy with task-specific requirements.
- Optimization and Validation
  The training process uses mixed precision training with `autocast` to reduce memory usage and improve efficiency on GPUs. Gradients are scaled using `GradScaler` to prevent underflow during backpropagation. After each epoch, the model is evaluated on validation data, tracking accuracy as the primary metric (correct predictions divided by total predictions). If validation accuracy improves, the model's state is saved as a checkpoint (`best_model.pth`). This ensures that the best-performing version of the model is retained for deployment or further fine-tuning. By freezing certain components like encoders and only training specific layers (e.g., fusion layer), this approach achieves efficient fine-tuning without overfitting or excessive resource usage.
- This code effectively combines visual and textual reasoning while optimizing performance through parameter-efficient techniques like mixed precision training and selective fine-tuning.

Hyperparamter tuning and maximum accuracy achieved:

The accuracy metric utilized for performance evaluation is accuracy- the correctly predicted true positive and true negative values versus the total values predicted. A maximum accuracy of 29.83% is achieved on test data and 33.42% is achieved on the validation data(taking into consideration pre

existing model like GAPS reaching around 68% accuracy(state of the art) and other models on an average accuracy of 45%).
100% of the training dataset has been utilized.
Some of the hyperparameters that have been tuned include:-

- Lower rank adaptation in loraconfig, for the phi-1.5 , it is 8. For bert based text encoder and EfficientB4net it is 16.
- The learning rate for different parameters of various models during training as specified:-

```
'params': phi_model.parameters(), "lr": 1e-5},
    {'params':image_encoder.parameters(),'lr':1e-5}, #learning rate for params determined via hyperparameter tuning
    {'params':text_encoder.parameters(),'lr':2e-5},
    {'params':fusion_layer.parameters(),'lr':3e-5}
```

- The test and validation split-70% and 30%
- The number of epochs=1
- Logits normalization(if normalized generalization on unseen data(cross validation accuracy reduces)

Issues faced and Potential Improvements:

- The first issue faced was training via lora and setting the target_modules, specially for Efficientb4Net Adapter- finally adjusted to the densely connected layers of the classifier head
- One major issue was resource constraint. Using better reasoning models like deepseek-math7b-instruct was impossible to train on ordinary cpus and even the ordinary T4 gpu as provided by colab. Settling down to a lightweight model was a necessity to reach the completion of training to begin with and to save the VRAM consumption while training on a simple T4 GPU.
- Another issue during the training was the continuity of the backpropagation chain, that is the classic “0 tensor does not have grad_fn” error, hence making sure every component even the decoder part (enabled via teacher forcing) did not detach from the backpropagation chain so that based on the predicted output all the

layers finetune-better image decoding, better text decoding, better fusion of both and better reasoning too- all learnt via training.

Potential Improvements:

- The potential improvements in the approach firstly would have been to actually implement inter-GPS, most efficiently using an end to end graph based training and to make the model even more scalable by building a dynamic knowledge graph, not only consisting of the existing theorems but using Retrieval Augmented Generation(RAG) on a real time basis to update the graph and at the same time applying a chain of thought mechanism.
- The simplest way to begin could have been to introduce a custom knowledge graph at first reinforced with existing mathematical(geometrical theorems specifically) and inducing a step by step solving approach(via chain of thought) to mimic the human way of solving math.
- Separate dataset training of the CNN on only geometry based data, against labelled values(supervised learning) since most CNNS are trained on general natural pictures(not specific to math)
- Symbolic analyser trained exclusively on math symbols to anyalize the mathematical expressions involved.
- Upgrading to higher GPU resources inevitably for better performance