SciGuru

Application Design Document

A. D. D

February, 2025

Table of content

Chapter 1 : Introduction

1.1 Summary

This research project focuses on training large language models (LLMs) to give better scientific explanations. By critically analyzing existing methodologies and leveraging state-of-the-art advancements, our goal is to fine-tune LLMs for improved precision and contextuality in scientific discourse.

Our approach integrates a blend of Supervised Fine Tuning (SFT) and different Reinforcement Learning (RL) methods, aiming to enhance the process of language modeling using up-to-date techniques.

We are creating and implementing an evaluation metric designed to measure the quality of scientific explanations. This metric enables us to gauge the success of our methods for enhancing the abilities of LLMs. Our goal is to fill a critical void regarding how LLMs function today, making them more useful for educational and research purposes. We're working towards enhancing how AI can help us understand and communicate scientific ideas, which could transform learning and discovery to be way easier in significant ways.

In addition to enhancing accuracy, we're placing significant emphasis on the quality of scientific explanations provided by LLMs. We aim to train a model that conveys complex scientific concepts in an engaging, clear, and contextually rich manner. Through this, we aspire to make science explanations more approachable and enjoyable for all.

1.2 Related Research & Scientific Background

Our research aims to enhance LLMs capability to generate high-quality scientific explanations. While LLMs have demonstrated remarkable abilities across various tasks, their proficiency in providing clear, accurate, and pedagogically sound scientific explanations can still be improved.

The field of scientific explanation assessment has been extensively researched, providing us with established frameworks to evaluate and enhance explanation quality. We reviewed several key papers to ensure our approach aligns with state-of-the-art methodologies in this domain.

1.2.1 TAP : "The Uses of Argument" (1958)

Toulmin's Argumentation Pattern (TAP), introduced by philosopher Stephen Toulmin, provides a structured approach to analyzing explanations through six key elements:

·	***Claim***	-	*The main conclusion being presented*
·	***Data***	-	*Supporting evidence and facts*
·	***Warrant***	-	*Logical connection between data and claim*
·	***Backing***	-	*Additional support for the warrant*
·	***Qualifier***	-	*Conditions under which the claim holds true*
·	***Rebuttal***	-	*Potential counterarguments or exceptions*

1.2.2 "An Instrument for Assessing Scientist's Written Skills in Public Communication of Science"

The article presents a comprehensive evaluation framework that identifies five critical components of effective scientific communication:

- **Content Features**: Focuses on the selection and organization of scientific information, including decisions about what to include or omit.
- **Knowledge Organization**: Examines how information is structured and reinforced through repetition and logical flow.
- **Analogical Approaches**: Evaluates the use of analogies and metaphors to explain complex concepts through familiar references.
- **Narrative Elements**: Assesses the incorporation of storytelling techniques to make scientific concepts more engaging and memorable.
- **Dialogic Communication**: Measures how well the explanation facilitates understanding and engagement, particularly for complex or controversial topics.

1.2.3 "Analyzing how Scientists Explain their Research"

Further insights come from the mentioned article, which presents rubric specifically designed for evaluating scientific explanations. This rubric emphasizes three key dimensions :

- **Pedagogical Knowledge -** Structure, Audience awareness, Language choice and technical presentation skills

- **Content Knowledge -** Factual accuracy, Information organization, Knowledge transfer capabilities

- **Integration -** Contextual placement, Visual and mental imagery, Scaffolded learning approach

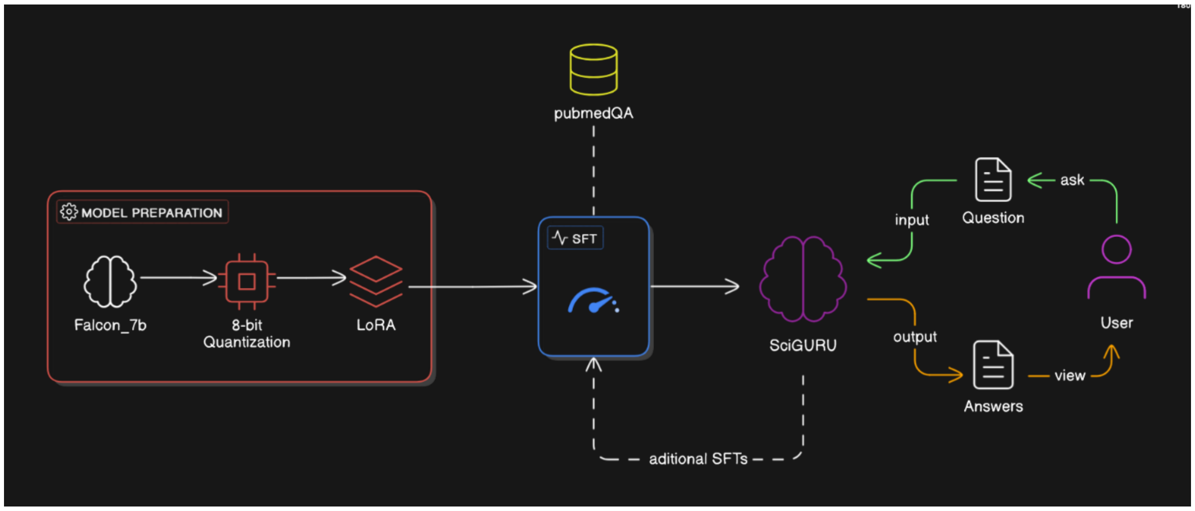
1.2.4 Summary

These frameworks provide our project with robust criteria for:

- Evaluating LLM-generated scientific explanations
- Developing effective training methodologies
- Creating comprehensive assessment metrics
- Guiding the fine-tuning process

By incorporating these established evaluation frameworks into our fine-tuning process, we aim to create a model that not only provides accurate scientific information but also presents it in a way that is pedagogically sound and accessible to diverse audiences. This approach ensures our work builds upon proven methodologies while advancing state-of-the-art AI-generated scientific explanations.

Chapter 2 : SciGuru Architecture



2.1 Supervised Fine Tuning (SFT)

The process of taking a **pre-trained LLM** and training it further on a **labeled dataset** *(meaning the data is carefully annotated with meaningful tags, or labels, that classify the data's elements or outcomes)* to align its outputs with specific tasks or objectives. By providing input-output examples (such as questions paired with correct answers), the model learns patterns and nuances that improve its performance on that domain. This method leverages the broad linguistic knowledge already present in the foundational model, while focusing on a particular use case, whether that's question answering, sentiment analysis, or any other NLP task.

2.1.1 Model Preparation

Model preparation is the process of readying a foundational language model for specialized tasks by optimizing its performance and efficiency. At a high level, this involves selecting a robust, pre-trained model and then applying techniques to streamline its computational demands without sacrificing quality. While the foundational model provides broad language understanding, methods like quantization and LoRA are later used to fine-tune the model's efficiency and adaptability. Essentially, model preparation ensures that the model is in the best possible shape to undergo further task-specific fine-tuning and deployment.

2.1.1.1 Base Model

Role in Fine-Tuning

Base model is a foundational, pre-trained Large Language Model (LLM) that has learned general language patterns from vast amounts of data. It serves as the starting point upon future fine-tuning.

Base Model in SciGuru

The chosen foundational model is a powerful and versatile LLM, known for its efficiency and performance on various NLP tasks. We selected the falcon model after experimenting with multiple other open-source models and continued with the falcon mamba due to its balancing of capability and efficiency, maximizing resources utilization over 7B parameters.

Other Base Models

T5_small (70M parameters)

A compact encoder-decoder model that we tested initially. While efficient in terms of computational resources, its limited parameter count resulted in less sophisticated scientific explanations compared to larger models. Due to its lack of parameters , T5_small was a perfect fit for initializing the finetuning pipeline without the need for a heavy computation but ultimately wasn't selected as our primary model for scientific inference.

T5_large (770M parameters)

A larger version of T5 that showed improved performance over T5_small but still fell short of the depth and nuance needed for complex scientific explanations. Similarly to T5_small this model is also a step stone towards the goal by experiencing work on the BGU cluster as well as integration of the project with GPU , that said, experimentation showed it's parameter count still low and doesn't fit with our expectations.

Llama_3.1 (8B parameters)

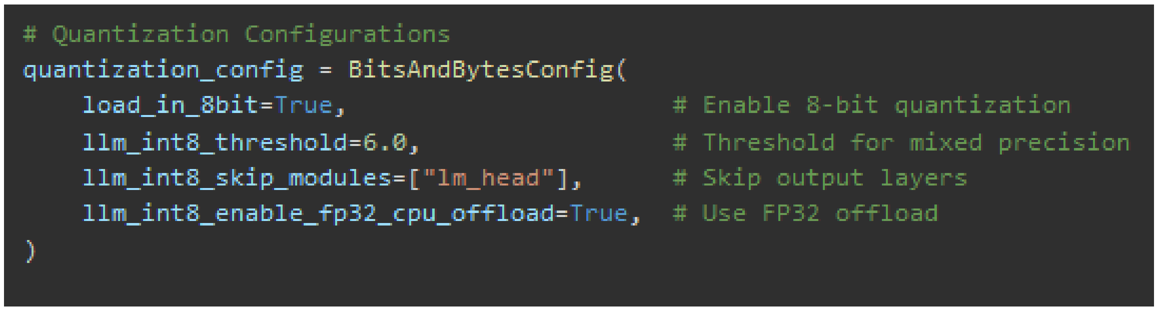
Meta's open-source model demonstrated strong performance but had licensing restrictions that could limit deployment options. While it's size and performance managed to fit our starting expectations, the deployment did not go smooth as we thought it would and the deadline was too close to waste more time, that said, **Llama_3.1** is a good contestant for **Falcon_7B** and we might use it in future versions.

2.1.1.2 Quantization

Quantization is used for optimization by reducing the memory and processing power required, it minimizes memory usage significantly and speeds inference, making the model more lightweight and suitable for integration into scalable pipelines.

Quantization in SciGuru

The Falcon-7b model undergoes 8-bit quantization, which reduces the model's precision to 8 bits while maintaining performance.



```
# Quantization Configurations
quantization_config = BitsAndBytesConfig(
    load_in_8bit=True,                      # Enable 8-bit quantization
    llm_int8_threshold=6.0,                 # Threshold for mixed precision
    llm_int8_skip_modules=["lm_head"],      # Skip output layers
    llm_int8_enable_fp32_cpu_offload=True,  # Use FP32 offload
)
```

load_in_8bit=True
Tells the library to load the model weights in 8-bit precision (instead of the usual 16-bit or 32-bit). This significantly reduces GPU memory usage and can speed up inference.

llm_int8_threshold=6.0
Sets the threshold for outlier detection in quantization. If a weight matrix has values whose absolute magnitude exceeds this threshold, that particular matrix is left in higher precision instead of being forced into 8-bit. This helps maintain accuracy for "outlier" values that would degrade performance if quantized.

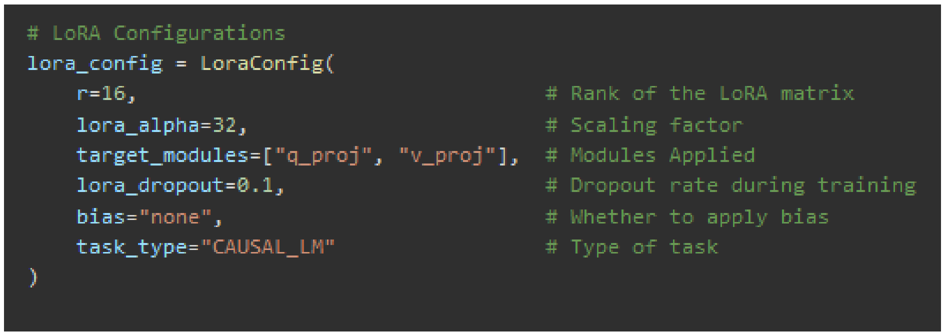
llm_int8_skip_modules=["lm_head"]
Specifies which modules to exclude from 8-bit quantization. In this case, the lm_head (the final output layer in a language model) is skipped and remains in full precision. This is often done to preserve output quality.

llm_int8_enable_fp32_cpu_offload=True
Enables offloading of certain calculations or parameters to CPU in 32-bit precision. This can help when GPU memory is limited, letting you keep some parameters in CPU RAM while still taking advantage of 8-bit quantization for the bulk of the model on the GPU.

2.1.1.3 LoRA

Low Rank Adaptation (LoRA) is applied to a model for efficient training and inference, instead of modifying the entire model parameters, LoRA adjusts smaller, low-rank matrices, reducing computational cost. LoRA allows the model to adapt to domain-specific tasks without needing extensive resources, ensuring efficient use of the pre-trained capabilities without causing catastrophic forgetting.

LoRA in SciGuru



```
# LoRA Configurations
lora_config = LoraConfig(
    r=16,                                  # Rank of the LoRA matrix
    lora_alpha=32,                         # Scaling factor
    target_modules=["q_proj", "v_proj"],   # Modules Applied
    lora_dropout=0.1,                      # Dropout rate during training
    bias="none",                           # Whether to apply bias
    task_type="CAUSAL_LM"                  # Type of task
)
```

r = 16
The rank of the low-rank matrices introduced by LoRA. This determines how many additional parameters get added during fine-tuning. A higher rank means more capacity (but also more parameters) for the LoRA adapters.

lora_alpha = 32
A scaling factor for the LoRA weights. Typically, the effective learning rate for LoRA parameters is multiplied by (lora_alpha / r). This helps control the update magnitude applied by the LoRA layers.

target_modules = ["q_proj", "v_proj"]
Specifies which model modules to apply LoRA to. In many transformer architectures, q_proj and v_proj are good choices because they significantly influence how attention is computed.

lora_dropout = 0.1
The dropout rate for LoRA's trainable params, can improve generalization and avoid overfitting.

bias = "none"
Determines how bias terms are handled in the LoRA layers. "none" indicates no additional bias terms are used for these adapters.

task_type = "CAUSAL_LM"
Specifies the task type as causal language modeling. This helps ensure that the fine-tuning process aligns with a left-to-right (causal) language model setup.

2.1.2 Data Preparation

Data preparation is the process of transforming raw text into a clean, structured, and consistent format that a model can effectively learn from. At a high level, this involves gathering and curating relevant data, cleaning it to remove noise or inconsistencies, and organizing it into a well-defined format. A key aspect of this process is tokenization—breaking down the text into units (words, subwords, or characters) that the model can understand. Essentially, effective data preparation lays the foundation for successful training by ensuring that the model receives high-quality, reliable input.

2.1.2.1 Picking The Right Dataset

Choosing the right dataset is the first crucial step in data preparation. It involves selecting a collection of texts that are not only relevant to your task but also representative of the domain you're targeting. For instance, if you're building a model for customer support, the dataset should include authentic customer interactions and responses. Factors such as data quality, diversity, and balance are essential - ensuring that the dataset has enough examples to capture the nuances of the language while avoiding biases. This careful curation lays a solid foundation, as the model's performance is highly dependent on the quality and relevance of the data it learns from.

Dataset in SciGURU

The dataset we chose is **PubMedQA**, a high-quality question-answering dataset derived from peer-reviewed biomedical literature. It provides expert-annotated Q&A pairs that help SciGURU develop an understanding of specialized terminology, structured reasoning, and fact-based answering. Since scientific texts are often dense, PubMedQA enables the model to simplify complex concepts while maintaining accuracy.

1. **Domain-Specific Knowledge** – It is sourced from credible scientific literature, ensuring the model learns from high-quality, peer-reviewed research.

2. **Fact-Based Question Answering** – The dataset emphasizes objective, evidence-based responses, which align with producing clear and accurate scientific explanations.

3. **Generalization Potential** – The structured Q&A format can be extended beyond biomedicine to other scientific fields, making it a versatile resource for simplifying complex concepts across multiple disciplines.

EXAMPLE

Question : Do preoperative statins reduce atrial fibrillation after coronary artery bypass grafting?

Context : [Abstract discussing a study on the effects of preoperative statin therapy on postoperative atrial fibrillation rates.]

Long Answer : [Conclusion stating that preoperative statin use is associated with a significant reduction in postoperative atrial fibrillation.]

Answer : Yes

2.1.2.2 Separation

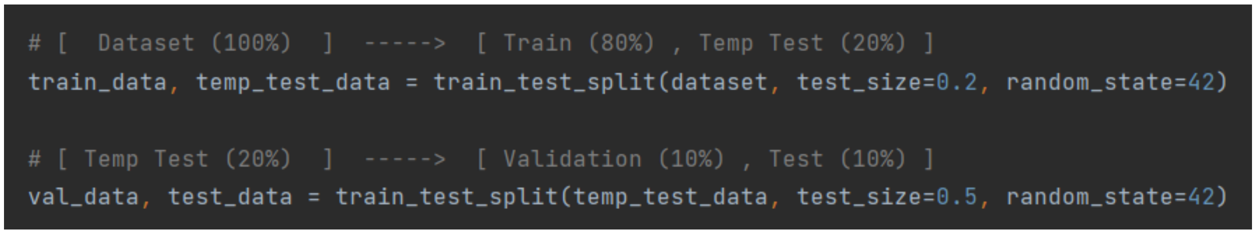
Separation refers to the process of splitting your dataset into distinct subsets, typically training, validation, and test sets. This division is crucial for developing a robust NLP model, and key to reliable model assessment and successful deployment.

Training Set - used for the actual learning process, where the model adjusts its parameters based on the provided examples.
Validation Set - helps monitor performance during training, allowing for hyperparameter tuning and early stopping to prevent overfitting.
Test Set - offers an unbiased evaluation of the model's performance on unseen data, ensuring that the model generalizes well to real-world scenarios.

Separation in SciGURU Dataset

In SciGURU we have split our dataset into training set, validation set and test set in the following ratio (randomly, to reduce chance for examples relativity – meaning that the data split example distribution among the dataset is random) :

Training Set	–	80% (~2900 examples)
Validation Set	–	10% (~360 examples)
Test Set	–	10% (~360 examples)



```
# [  Dataset (100%)  ]  ----->  [ Train (80%) , Temp Test (20%) ]
train_data, temp_test_data = train_test_split(dataset, test_size=0.2, random_state=42)

# [ Temp Test (20%)  ]  ----->  [ Validation (10%) , Test (10%) ]
val_data, test_data = train_test_split(temp_test_data, test_size=0.5, random_state=42)
```

2.1.2.3 Pre-processing

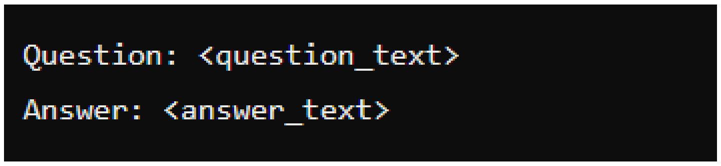
Transforms raw text into a cleaner, more consistent format, making it suitable for training.
Preprocessing typically includes:

- **Cleaning the text** (removing special characters, punctuation, numbers, extra spaces, etc.)
- **Lowercasing** (optional, depending on the task)
- **Removing stop words** (for some tasks, but not always necessary)
- **Stemming/Lemmatization** (reducing words to their root form)

The goal of preprocessing is to eliminate inconsistencies and extraneous information that could confuse the model, ensuring that only valuable, structured information is fed into the system. Additionally, it often includes splitting the dataset into training, validation, and test sets to help evaluate model performance effectively.

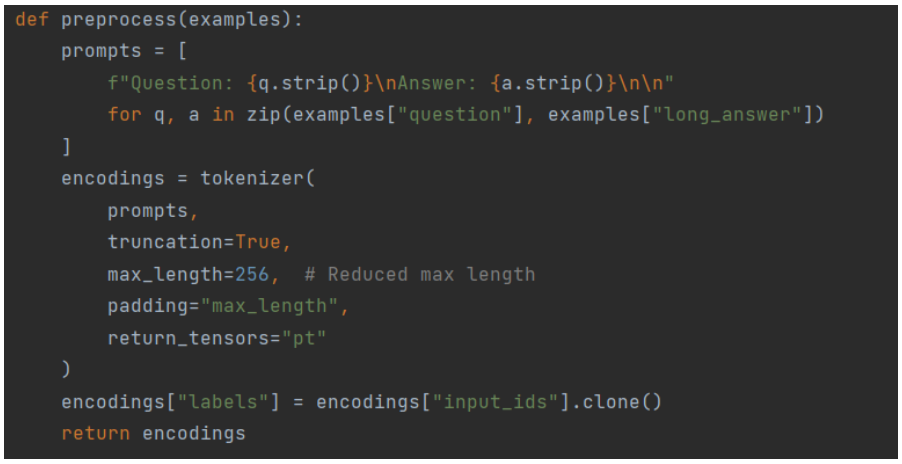
-

- Extracted **question-answer pairs** from the dataset.
- Formated them into a structured text prompt



```
Question: <question_text>
Answer: <answer_text>
```

- Used .strip() to remove extra spaces.
- Added \n\n for better separation and clarity.
- Processed data in **batches** using dataset.map().
- Removed original dataset columns after processing.



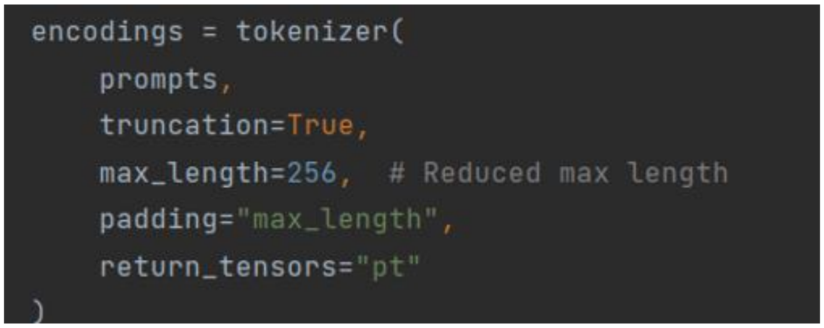
```python
def preprocess(examples):
    prompts = [
        f"Question: {q.strip()}\nAnswer: {a.strip()}\n\n"
        for q, a in zip(examples["question"], examples["long_answer"])
    ]
    encodings = tokenizer(
        prompts,
        truncation=True,
        max_length=256,  # Reduced max length
        padding="max_length",
        return_tensors="pt"
    )
    encodings["labels"] = encodings["input_ids"].clone()
    return encodings
```

2.1.2.4 Tokenization

Tokenization is the critical process of breaking down text into smaller units called tokens, which are the basic elements that a model understands. Depending on the approach, tokens can be words, sub words, or even characters. Modern NLP methods often favor sub word tokenization techniques, which strike a balance between handling rare words and managing vocabulary size efficiently. Tokenization converts the cleaned text into numerical representations - typically sequences of integers that correspond to tokens in the model's vocabulary - thereby bridging the gap between human-readable language and machine-readable data. This conversion is fundamental for model training, as it directly influences how the model perceives and processes the language.

Tokenization in SciGuru

- Used truncation (truncation=True) to limit sequences to 256 tokens.
- Applied padding (padding="max_length") to standardize input size.
- Converted the processed tokens into PyTorch tensors (return_tensors="pt").
- Duplicated input_ids as labels, meaning the model learns to predict the next token.
- Ensured the processed dataset is ready for model training.



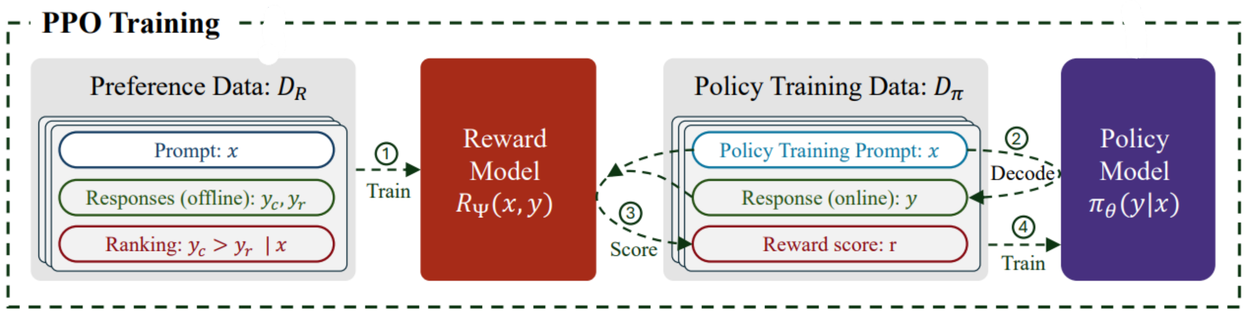
```
encodings = tokenizer(
    prompts,
    truncation=True,
    max_length=256,  # Reduced max length
    padding="max_length",
    return_tensors="pt"
)
```

2.2 Reinforcement Learning (RL)

Reinforcement learning is a process where an agent learns optimal behavior through repeated interactions with its environment. In the context of LLM's, there exists a range of techniques used to train a model, where each prompt describes an action, and the model is given feedback for its actions in the form of a scalar reward. Repeating this process helps the agent learn which actions to take to maximize its overall reward. One of our objectives in this project is to apply 4 different RL techniques to train the model and evaluate their effect on the explainability of the model. The techniques we will apply are Direct Policy Optimization (DPO), Proximal Policy Optimization (PPO), Reinforcement Learning with Verifiable Rewards (RLVR), and classic reinforcement learning using ChatGPT's API- which we can refer to as classic RL.

2.2.1 Proximal Policy Optimization (PPO)

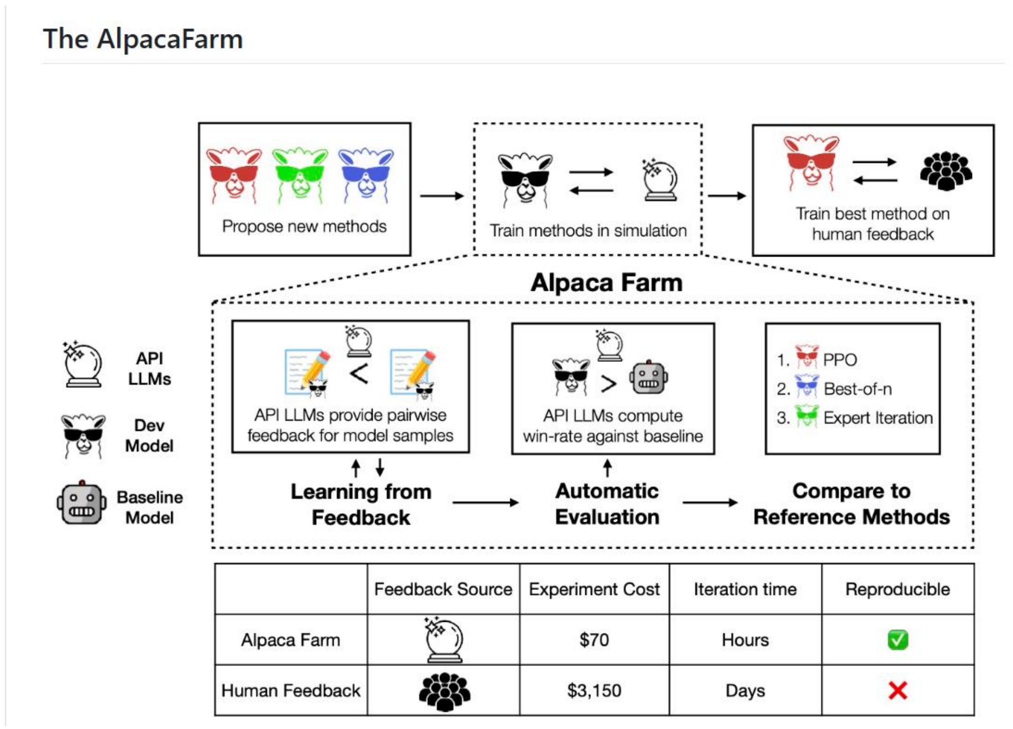
PPO is the process of learning from what is often refered to as a *preference dataset*. A preference dataset is derived from question (prompts) and pairs of answers. In a preference dataset for each question x there are 2 labeled answers - y_c, y_r .
y_c is labelled as the chosen (preferred) answer, and y_r is labelled as the rejected answer. In PPO, the process consists of first training the reward model on the preference dataset – and then training a policy model. The training of the policy is given prompts, for which the model produces answers which are scored using the reward model.


PPO Training
Preference Data: D_R
Prompt: x
Responses (offline): y_c, y_r
Ranking: $y_c > y_r \mid x$
① Train
Reward Model $R_\Psi(x, y)$
③ Score
Policy Training Data: D_π
Policy Training Prompt: x
Response (online): y
Reward score: r
② Decode
④ Train
Policy Model $\pi_\theta(y|x)$

The process of producing responses and receiving rewards to improve a policy is often referred to as 'online' training.

2.2.1.1 PPO Preference Dataset

To develop an effective reward model, we require a dataset containing paired responses with clear quality rankings. While we have access to the *PubMedQA* questions, we need a systematic way to generate and evaluate multiple answers for each question. Manual evaluation of these answer pairs would be impractical given our team size and resource constraints. Instead, we will utilize the AlpacaFarm framework, which provides a structured approach to generating and evaluating responses. This framework leverages existing language models and reference responses to create a scalable system for producing and ranking answer pairs, effectively simulating human preference judgments.


The AlpacaFarm

Propose new methods
Train methods in simulation
Train best method on human feedback
Alpaca Farm
API LLMs
Dev Model
Baseline Model
API LLMs provide pairwise feedback for model samples
API LLMs compute win-rate against baseline
1. PPO
2. Best-of-n
3. Expert Iteration
Learning from Feedback
Automatic Evaluation
Compare to Reference Methods

	Feedback Source	Experiment Cost	Iteration time	Reproducible
Alpaca Farm		$70	Hours	✅
Human Feedback		$3,150	Days	❌

In our case, prompts would be only the questions from the PubMedQA, our LLM model would be the baseline model and ChatGPT's API would be used to rank and give feedback to the pairs of responses.

2.2.2 Direct Policy Optimization (DPO)

While PPO includes producing responses and training a policy in what is called an "online process", DPO employs an offline approach that learns directly from existing preference data. Rather than constructing a separate reward model or generating new samples during training, DPO optimizes its policy by maximizing the difference between the probability scores of preferred responses and rejected ones. This streamlined approach eliminates the need for intermediate reward modeling and iterative sampling steps.

The main difference between the methods can be illustrated in the following figure:

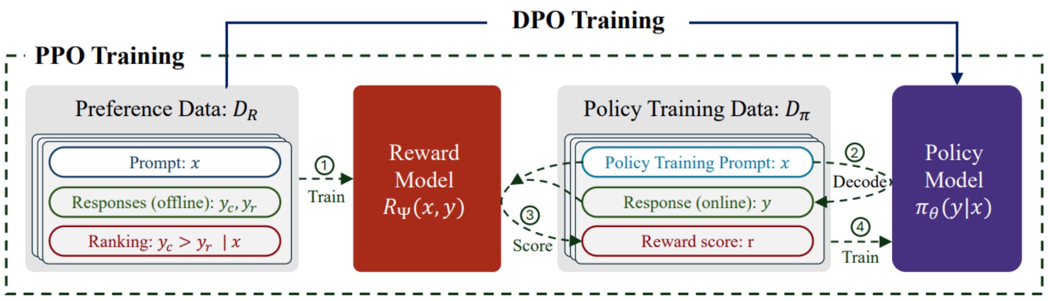
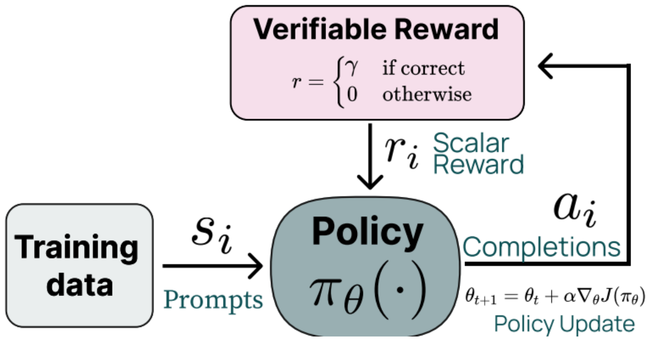

DPO Training
PPO Training
Preference Data: D_R
Prompt: x
Responses (offline): y_c, y_r
Ranking: $y_c > y_r \mid x$
① Train
Reward Model $R_\Psi(x, y)$
③ Score
Policy Training Data: D_π
Policy Training Prompt: x
Response (online): y
Reward score: r
② Decode
④ Train
Policy Model $\pi_\theta(y|x)$

Figure 2: The core aspects of learning from preference feedback. For DPO (solid line), preference data is directly used to train a policy model. For PPO (dashed line), preference data is used to train a reward model, which is then used to score model-generated responses during PPO training.

Both DPO and PPO will be trained using the same dataset of preferred and non-preferred responses. This allows us to make a fair comparison between the two methods. We will base our implementation on recent research papers, specifically "Direct Preference Optimization: Your Language Model is Secretly a Reward Model" and "Unpacking DPO and PPO," along with other key studies listed in our references. This approach ensures our comparison follows established best practices while maintaining consistent testing conditions.

2.2.3 Reinforcement Learning with Verifiable Rewards (RLVR)

In RLVR rewards are given if an answer meets a given verifiable condition. For example - given a mathematical question, an answer is only rewarded if it is mathematically correct. We aim to define a set of criteria/metrics to verify different aspects in our model's responses. Our main objective here is to implement these assessments of desirable behavior, and grant rewards per each fulfilled aspect.


Verifiable Reward
$r = \begin{cases} \gamma & \text{if correct} \\ 0 & \text{otherwise} \end{cases}$
r_i Scalar Reward
Training data
s_i
Prompts
Policy
$\pi_\theta(\cdot)$
a_i
Completions
$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\pi_\theta)$
Policy Update

The figure source is the Tülu 3: Pushing Frontiers in Open Language Model Post-Training paper, which we will use a reference to implement our work.

2.2.3.1 RLVR Reward Model

Our initial set of criteria is listed in *"chapter 1 – related research"*, and for each of the criteria we shall use ChatGPT's API to verify whether the generated answers hold to the predefined standard, as we hope to iterate over thousands of examples and will only be able to do so via an automated process. We also consider using DeepSeek's API which will allow us to run 27x examples with the same budget.

2.2.3.2 RLVR Training Data

As RLVR doesn’t require references, only deterministically verifiable outputs – we will use the *PubMedQA* questions for prompts, and the rewards will be given to the answers generated by the model, if they match the required output. We will use the same questions we used for the training and testing for the other methods, neglecting the references similarity evaluation and generating rewards solely on desirable behavior.

2.2.4 RL with ChatGPT API

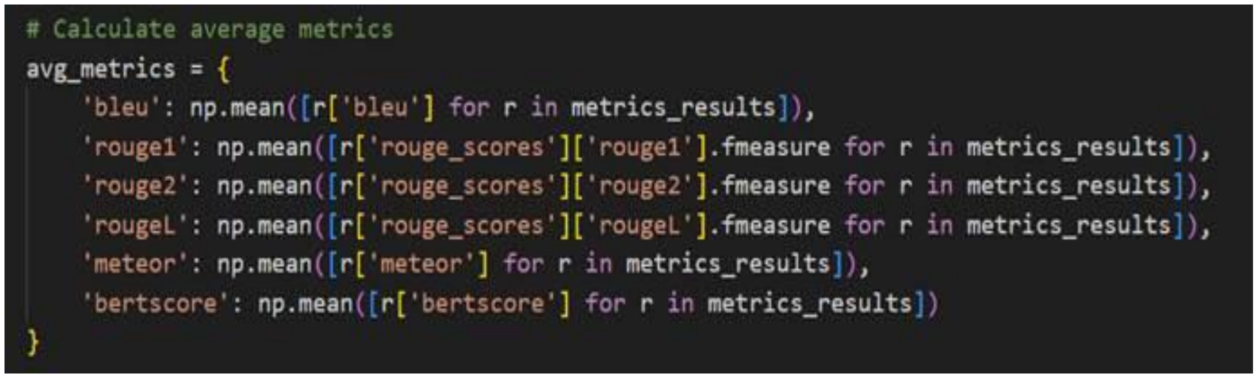
In the last method we will use the ChatGPT API to provide rewards to responses. The prompts we will use will ask for ChatGPT to rate the explainablity of the answers and give one reward per answer. We will use the same training data as with RLVR. This method should replicate the judgement of a human for a well explained answer. The API acts the reward model - the policy update and training process is identical to RLVR.

Chapter 3 : Evaluation

3.1 Evaluation Metrics

As mentioned before, the dataset was separated into Training, Validation and test sets. In order to evaluate the model performance after training (meaning that the model has only "seen" Training and Validation sets), we have to use the test set (~3000 Q&A which the model has never seen or trained on) to test our current alpha model, the model responded to each question and its answer was compared to a suitable reference answer using predetermined quality metrics.

Quality Metrics



```
# Calculate average metrics
avg_metrics = {
    'bleu': np.mean([r['bleu'] for r in metrics_results]),
    'rouge1': np.mean([r['rouge_scores']['rouge1'].fmeasure for r in metrics_results]),
    'rouge2': np.mean([r['rouge_scores']['rouge2'].fmeasure for r in metrics_results]),
    'rougeL': np.mean([r['rouge_scores']['rougeL'].fmeasure for r in metrics_results]),
    'meteor': np.mean([r['meteor'] for r in metrics_results]),
    'bertscore': np.mean([r['bertscore'] for r in metrics_results])
}
```

BLEU (Bilingual Evaluation Understudy):

Measures how similar a generated text is to a reference text using n-gram overlap.
Higher BLEU means better similarity to the reference text.
Often used in machine translation.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

Measures text overlap, especially useful for summarization tasks, when higher scores mean the generated text captures more key parts of the reference text.

ROUGE-1 Measures overlap of single words (unigrams).
ROUGE-2 Measures overlap of word pairs (bigrams).
ROUGE-L Measures longest common subsequence (captures fluency and coherence).

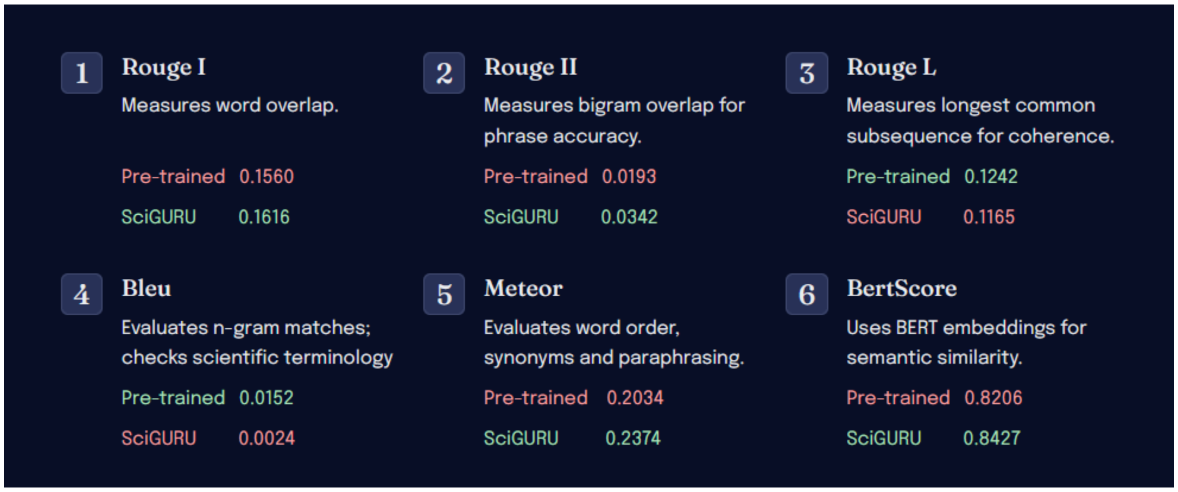
METEOR (Metric for Evaluation of Translation with Explicit Ordering)

Improves BLEU by considering synonyms, stemming and word order, when a higher METEOR score means better alignment with the reference text.

BERTScore

Uses deep learning (BERT) to compare similarity at the meaning level, not just word overlap. Captures semantic similarity between words in context, when a higher BERTScore means the generated text conveys a similar meaning to the reference.

Evaluation Results



	Metric	Description	Pre-trained	SciGURU
1	Rouge I	Measures word overlap.	0.1560	0.1616
2	Rouge II	Measures bigram overlap for phrase accuracy.	0.0193	0.0342
3	Rouge L	Measures longest common subsequence for coherence.	0.1242	0.1165
4	Bleu	Evaluates n-gram matches; checks scientific terminology	0.0152	0.0024
5	Meteor	Evaluates word order, synonyms and paraphrasing.	0.2034	0.2374
6	BertScore	Uses BERT embeddings for semantic similarity.	0.8206	0.8427

3.2 RL Evaluation – ranking the models

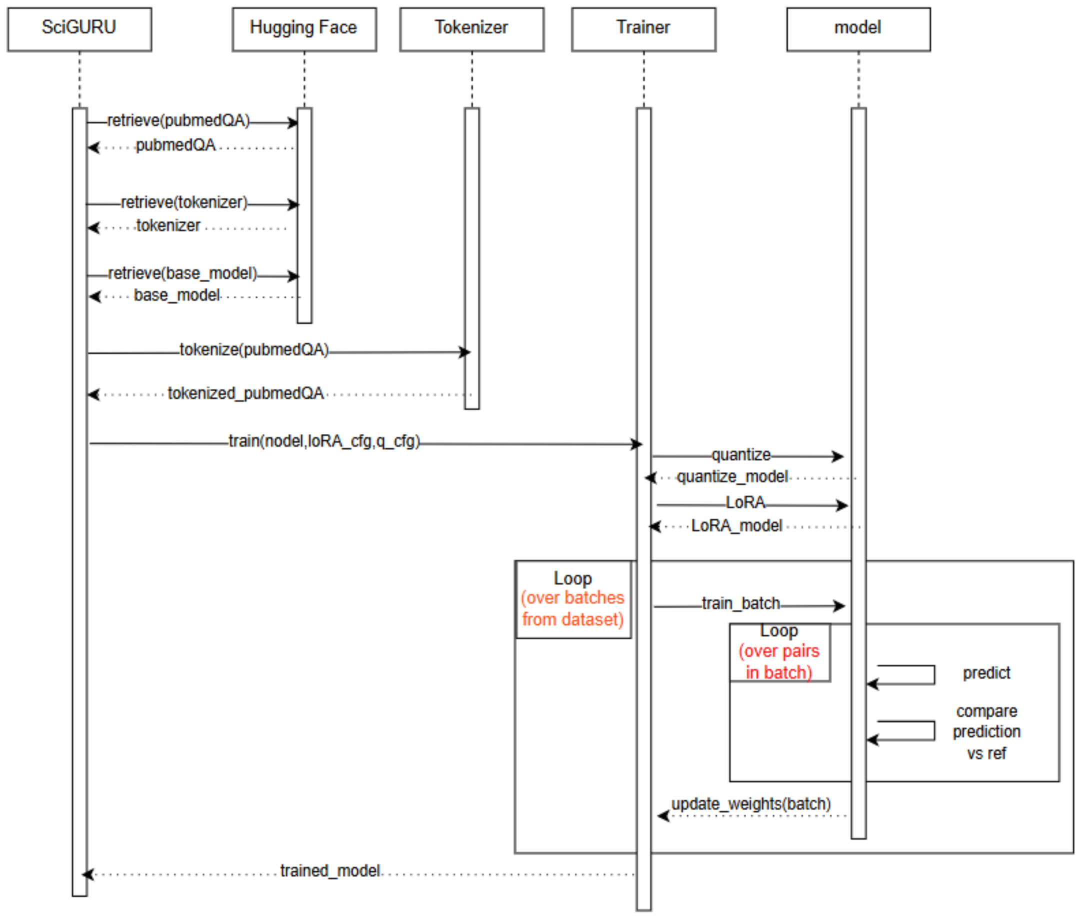
Our evaluation framework employs both automated and manual assessment methods to compare the performance of different model implementations. The evaluation will be conducted across six models: the base model, fine-tuned model, and four RL-based implementations. Using our test dataset of 360 examples, we will assess the explainability of responses through two approaches:

1) Automated Assessment:
   - 300 responses will be evaluated using the ChatGPT API
   - Each response will receive a ranking from 1 (highest) to 6 (lowest)
   - The API will follow standardized criteria based on established research benchmarks
2) Human Verification:
   - 60 responses will undergo manual review
   - This provides a critical validation check against the automated assessments
   - Uses the same 1-6 ranking system for consistency
3) Reward Model Validation:
   - Calculate the cumulative rewards for each RL model's responses
   - Compare the reward rankings with the ChatGPT rankings
   - If the model with the highest rewards also achieves the best rankings from ChatGPT, this validates our reward metrics
   - This correlation would confirm that our reward models effectively capture genuine response quality

Chapter 4: Behavioral Analysis

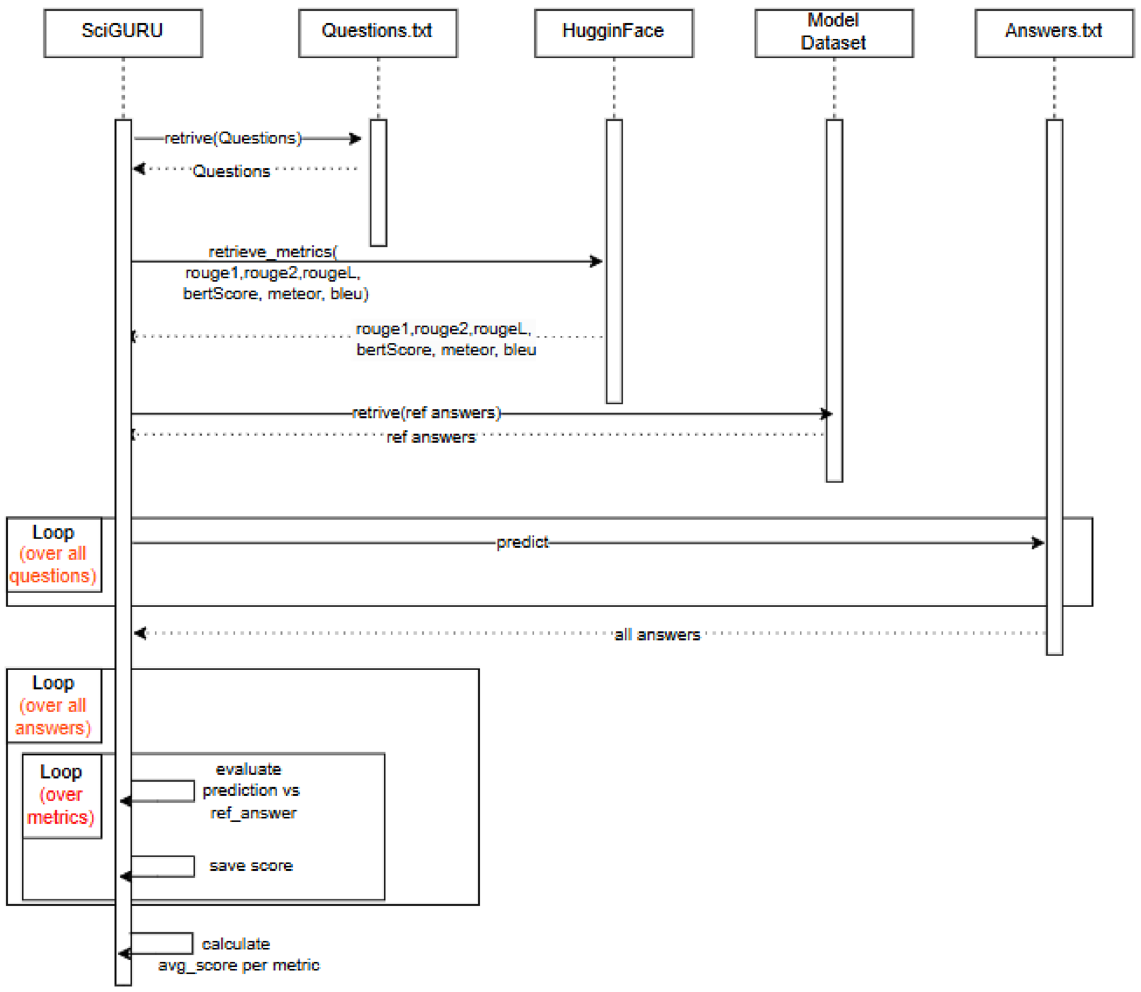
4.1 Sequence Diagrams

4.1.1 Training:



The diagram illustrates our SFT training pipeline. We begin with SciGuru, the main orchestrator, retrieving the PubMedQA dataset, tokenizer, and base model from Hugging Face. After tokenizing the PubMedQA data, SciGURU initiates our training process with specific LoRA configurations. Then, the Trainer component applies 4-bit quantization to the model and implements the LoRA adaptations. The core training loop processes batches of data, where the model makes predictions and compares them against our reference answers. After each batch, the weights are updated accordingly. This iterative process continues for all pairs within each batch until the training is complete, ultimately producing a trained model that's returned to SciGURU.

4.1.2 Evaluation:


SciGURU
Questions.txt
HugginFace
Model Dataset
Answers.txt
retrive(Questions)
Questions
retrieve_metrics(rouge1,rouge2,rougeL, bertScore, meteor, bleu)
rouge1,rouge2,rougeL, bertScore, meteor, bleu
retrive(ref answers)
ref answers
Loop (over all questions)
predict
all answers
Loop (over all answers)
Loop (over metrics)
evaluate prediction vs ref_answer
save score
calculate avg_score per metric

The diagram depicts the evaluation process for assessing the trained model's explanation quality. The flow begins with SciGURU retrieving test questions from a Questions.txt file, followed by loading multiple evaluation metrics including ROUGE-1, ROUGE-2, ROUGE-L, BERT Score, METEOR, and BLEU from Hugging Face. SciGURU then retrieves reference answers from our Model's Dataset. The process enters three loops: first, it iterates through all questions, generating predictions which are stored in Answers.txt. Then, all answers are processed, and the innermost loop evaluates each metric by comparing predictions against reference answers. Finally, scores are saved and averaged per metric to produce the final evaluation results. This evaluation framework allows us a systematic assessment of the model's performance in generating scientific explanations across multiple quality dimensions.

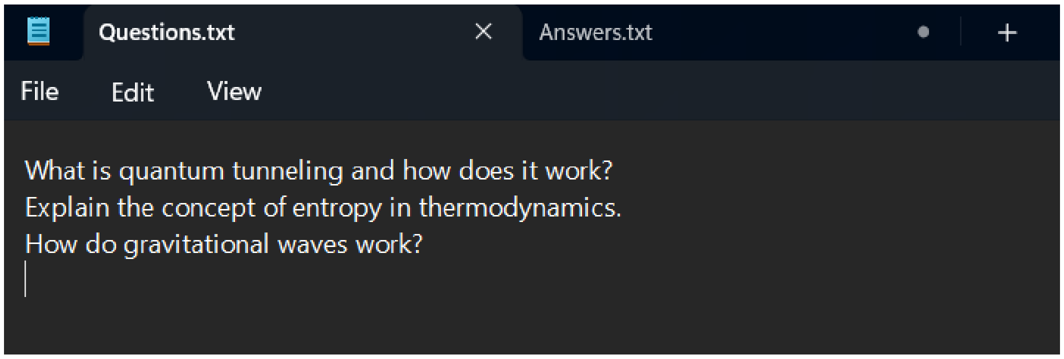
Chapter 5 : User Interface

Since training and evaluating the models is computationaly complex, the only way to interact with our model is using using SSH connection to the university's SLURM cluster where we run the model's training and evaluation.

For the alpha version we currently have 2 text files on the cluster: Questions.txt and Answers.txt.

To ask the model a question (or several questions) the user must follow the following steps:

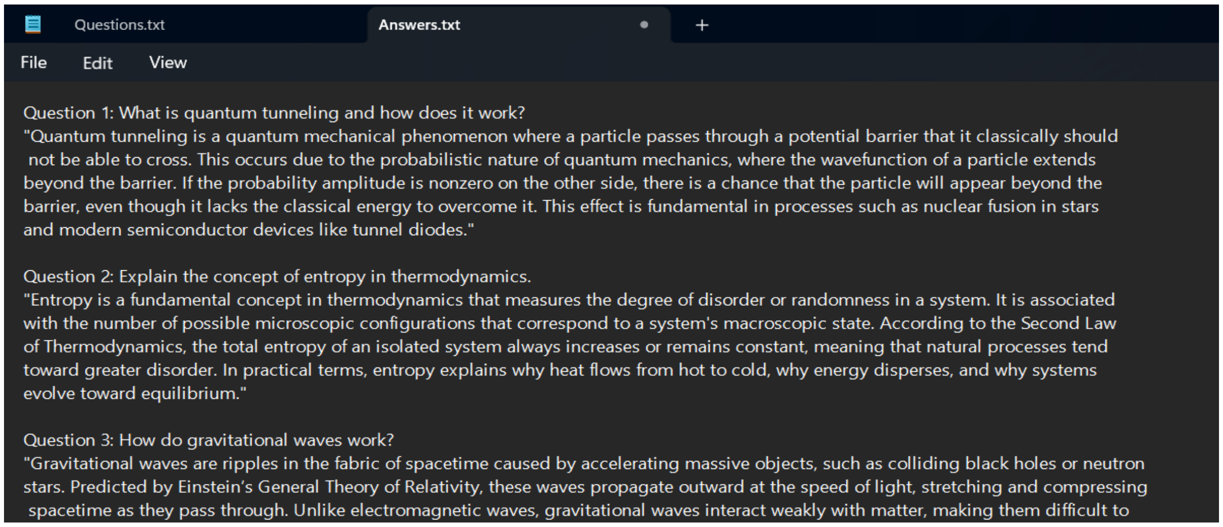
1. Open Questions.txt file and type in the questions that they wish to ask SciGURU.



```
What is quantum tunneling and how does it work?
Explain the concept of entropy in thermodynamics.
How do gravitational waves work?
```

2. Run a sbatch script, which runs SciGURU inference on Questions.txt and outputs to Answers.txt.

3. Open Answers.txt to watch the model's response

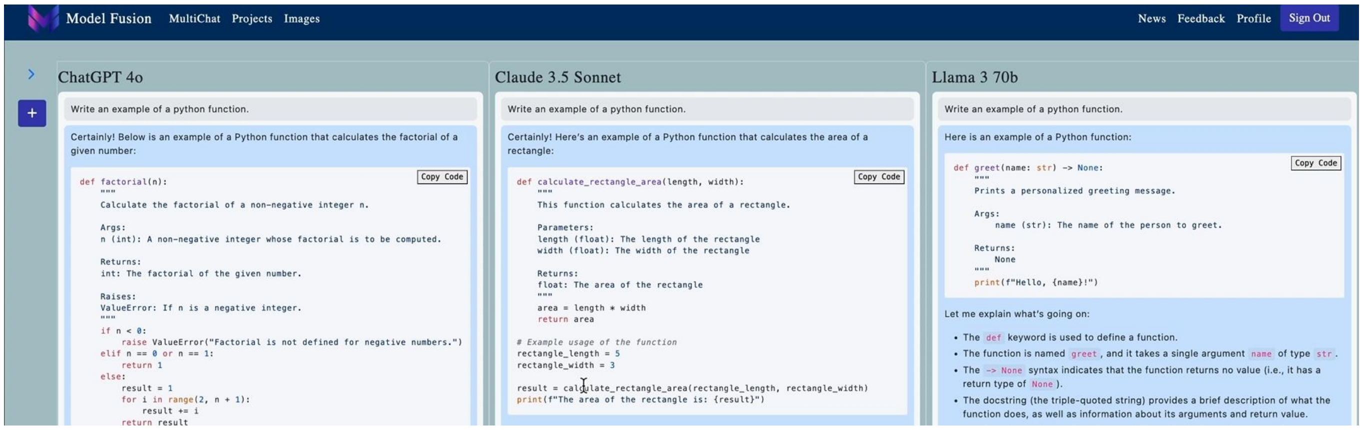


```
Question 1: What is quantum tunneling and how does it work?
"Quantum tunneling is a quantum mechanical phenomenon where a particle passes through a potential barrier that it classically should
 not be able to cross. This occurs due to the probabilistic nature of quantum mechanics, where the wavefunction of a particle extends
beyond the barrier. If the probability amplitude is nonzero on the other side, there is a chance that the particle will appear beyond the
barrier, even though it lacks the classical energy to overcome it. This effect is fundamental in processes such as nuclear fusion in stars
and modern semiconductor devices like tunnel diodes."

Question 2: Explain the concept of entropy in thermodynamics.
"Entropy is a fundamental concept in thermodynamics that measures the degree of disorder or randomness in a system. It is associated
with the number of possible microscopic configurations that correspond to a system's macroscopic state. According to the Second Law
of Thermodynamics, the total entropy of an isolated system always increases or remains constant, meaning that natural processes tend
toward greater disorder. In practical terms, entropy explains why heat flows from hot to cold, why energy disperses, and why systems
evolve toward equilibrium."

Question 3: How do gravitational waves work?
"Gravitational waves are ripples in the fabric of spacetime caused by accelerating massive objects, such as colliding black holes or neutron
stars. Predicted by Einstein's General Theory of Relativity, these waves propagate outward at the speed of light, stretching and compressing
 spacetime as they pass through. Unlike electromagnetic waves, gravitational waves interact weakly with matter, making them difficult to
```

Using Gradeio, in the final version we plan to build basic UI application which will run locally and present the different responses of each of the trained models. We expect the UI to look something like this:



Write an example of a python function.

Certainly! Below is an example of a Python function that calculates the factorial of a given number:

```
def factorial(n):
    """
    Calculate the factorial of a non-negative integer n.

    Args:
    n (int): A non-negative integer whose factorial is to be computed.

    Returns:
    int: The factorial of the given number.

    Raises:
    ValueError: If n is a negative integer.
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result += i
        return result
```

Write an example of a python function.

Certainly! Here's an example of a Python function that calculates the area of a rectangle:

```
def calculate_rectangle_area(length, width):
    """
    This function calculates the area of a rectangle.

    Parameters:
    length (float): The length of the rectangle
    width (float): The width of the rectangle

    Returns:
    float: The area of the rectangle
    """
    area = length * width
    return area

# Example usage of the function
rectangle_length = 5
rectangle_width = 3

result = calculate_rectangle_area(rectangle_length, rectangle_width)
print(f"The area of the rectangle is: {result}")
```

Write an example of a python function.

Here is an example of a Python function:

```
def greet(name: str) -> None:
    """
    Prints a personalized greeting message.

    Args:
        name (str): The name of the person to greet.

    Returns:
        None
    """
    print(f"Hello, {name}!")
```

Let me explain what's going on:

- The `def` keyword is used to define a function.
- The function is named `greet`, and it takes a single argument `name` of type `str`.
- The `-> None` syntax indicates that the function returns no value (i.e., it has a return type of `None`).
- The docstring (the triple-quoted string) provides a brief description of what the function does, as well as information about its arguments and return value.

Chapter 6 : Infrastructure

SciGURU is built on a robust computational infrastructure designed for efficient execution and development. The project operates on a **cluster-based system**, leveraging **RTX 4090 / RTX 3090 GPUs** (one per sbatch script) to manage workload distribution effectively. The development workflow follows the **SCRUM methodology**, ensuring iterative improvements and streamlined collaboration.

6.1 Resources

Computational Infrastructure

- Runs on a high-performance computing cluster with allocated **RTX 4090 / 3090 GPUs**.
- Workload distribution and model training are managed using **sbatch scripts**

Execution and Deployment

- Fully script-based execution model.
- Automated data processing, training, and evaluation workflows.
- Logging and monitoring ensure stable and trackable model iterations.

6.2 Directory Architecture

Log

- Stores all runtime logs, capturing details of model training, execution, and debug information.
- Logs are structured for easy tracing of issues and performance evaluation.
- Helps maintain a history of all major training runs and parameter configurations.

Utils

- API keys and authentication tokens.
- Environment setup files (requirements.txt, .env configurations).
- "*Helper*" scripts for debugging and monitoring system performance.

Versions

- Contains all trained versions of SciGuru, allowing for comparison and iteration.
- Includes metadata on hyperparameters and training configurations for reproducibility.
- Ensures past versions can be reloaded and used for further fine-tuning or evaluation.

Finetuning

- Houses datasets, python files related to the fine-tuning process.
- Supports different models and dataset variations for experimentation.

Scripts

- Fine-tuning the model using predefined training configurations.
- Resource loading, loading hugging face models, tokenizers and datasets.
- Environment setup for configuring dependencies and initializing the system.
- Environment reset scripts for restoring a clean working state when needed.

By structuring the project in this way, SciGURU ensures **scalability, efficiency, and organization**, facilitating seamless collaboration and continuous improvements in model performance.

Bibliography

Osborne, J., Erduran, S., & Simon, S. (2004). Enhancing the quality of argumentation in school science. Journal of Research in Science Teaching, 41(10), 994–1020. https://doi.org/10.1002/tea.20035

Toulmin, S. E. (2003). The Uses of Argument (2nd ed.). Cambridge: Cambridge University Press.

Baram-Tsabari, A., & Lewenstein, B. V. (2013). An Instrument for Assessing Scientists' Written Skills in Public Communication of Science. Science Communication, 35(1), 56–85. https://doi.org/10.1177/1075547012440634

Sevian, H., & Gonsalves, L. (2008). Analysing how Scientists Explain their Research: A rubric for measuring the effectiveness of scientific explanations. International Journal of Science Education, 30(11), 1441–1467. https://doi.org/10.1080/09500690802267579

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., ... & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems. https://arxiv.org/abs/2307.09288

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., ... & Lowe, R. (2022). Training language models to follow instructions with human feedback. Advances in Neural Information Processing Systems, 35, 27730-27744. arXiv:2203.02155

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., ... & Kaplan, J. (2022). Constitutional ai: Harmlessness from ai feedback. arXiv preprint arXiv:2212.08073.

Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017)

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., & Iwasawa, Y. (2022). Large language models are zero-shot reasoners. Advances in neural information processing systems, 35, 22199-22213. arXiv:2205.11916

Unpacking DPO and PPO: Disentangling Best Practices for Learning from Preference Feedback (2024). arXiv preprint https://arxiv.org/abs/2406.09279.