CSI – 5130(Artificial Intelligence)

Adaptive Prompt Optimization using Reinforcement Learning (RLHF-lite)

Presented By:
Agna Antony
Vineela Rao Akasapu
Mubassir Gamfoi

OAKLAND UNIVERSITY

Agenda

- Introduction
- Background On Large Language Models
- Reinforcement Learning
- RLHF-Lite Framework for Prompt Optimization
- System Design and Components
- Techniques
- Implementation and Training Process
- Results
- Conclusion
- Challenges and Future work
- References

Introduction - Adaptive Prompt Optimization

Large language models (LLMs) are now widely used for many NLP tasks, but their outputs are highly sensitive to how the input prompt is written.

Their behavior is highly prompt dependent: well designed prompts improve clarity and correctness, while poor prompts cause irrelevant, verbose, or inaccurate outputs.

Manual prompt engineering is subjective, slow, and hard to scale because it relies on expert intuition and repeated trial and error.

RLHF has shown that reinforcement learning can align LLMs with human preferences, but it typically needs costly human feedback, reward models, and significant compute.

Adaptive prompt optimization aims to automate this process by treating prompt selection as a learning problem, where an agent iteratively improves prompts based on feedback.

Background - Large Language Models

- Large Language Models like GPT-4, Mistral, LLaMA and T5 enable tasks such as summarization, translation, reasoning and dialogue with near human fluency.

- Built on the Transformer architecture, which uses self attention to model long range dependencies and rich contextual relationships in text.

- Transformers process all tokens in parallel instead of word by word, allowing them to capture subtle linguistic patterns and global context.

- LLMs are sequence to sequence or autoregressive generators that produce text one token at a time from a learned probability distribution.

- Because of this design, their behavior is highly sensitive to the structure, clarity and intent of the input prompt, making prompt design a critical factor in output quality.

Exploring the Google Flan-T5 Model

- **Overview of flan-t5:**

  By transforming all tasks into a single text-to-text format, flan-t5, a cutting-edg text-to-text transformer model, performs exceptionally well in arrange of natural language processing tasks. Because of its adaptability, it can handle a wide range of tasks with exceptional accuracy and efficiency, including translation and summarization.

- **Advantage of flan-t5:**

  The model's architecture makes good use of transfer learning, allowing it to perform well in a variety of tasks. Its pre-training on a sizable corpus of varied data improves its comprehension of linguistic context and subtleties, making it especially well-suited for adaptive prompt optimization.

Workflow – RLHF Lite

Dataset Preparation
- Load text passages & reference summaries

Load FLAN-T5 Base
- Use as frozen target LLM

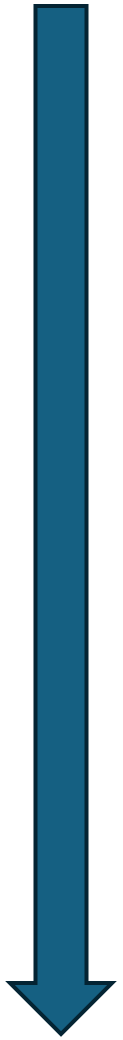
Define Template Pool
- Create candidate prompts

RL Loop
- Run Q-learning for 50 iterations

Reward Function
- Use ROUGE+BLEU

Identify Optimized Prompt
- Select highest-Q template

Evaluate Before vs After
- Compare rewards on test inputs



Prompt Engineering for LLMs

- Prompt engineering is the process of crafting and refining input instructions to elicit desired behavior from an LLM.
- Effective prompts balance specificity, clarity and stylistic cues to improve coherence, correctness and relevance.
- Small wording changes can dramatically affect outputs, for example:
  - “TL;DR:” vs “Summarize the following text”
  - “Summarize in a single sentence”
  - “What are the main points?
- This sensitivity arises because LLMs rely on patterns learned from large corpora where certain phrases act as strong semantic triggers.
- As a result, manual prompt engineering is often subjective, iterative and hard to scale across different tasks and datasets.

RLHF-Lite for Prompt Optimization

Q-Learning Prompt Optimization
Reward Function based ROUGE, BLEU
Use of key libraries (Transformers , Torch etc.)

Reinforcement Learning

- In the machine learning paradigm known as Reinforcement learning, agents learn to make decisions by optimizing their actions to maximize cumulative rewards after receiving feedback from their surroundings.

- **State, action, and reward in LLM context**

  The "action" in LLMs is choosing a prompt or response strategy, the "state" is the current prompt and model output, and the "reward" is the assessment of the output quality using predetermined metrics.

Q-Learning

The space of potential prompts is explored by a Q-learning agent, which learns from previous interactions to choose the best prompts for producing high-quality outputs.

Learning rule (Q-learning):

The agent updates the value of each prompt template using $Q[a] = Q[a] + \alpha \times (\text{reward} - Q[a])$ with $\alpha = 0.2$ and epsilon-greedy exploration, gradually favoring the prompts that produce better summaries.

Exploration Vs Exploitation :

Finding the right balance between exploration (trying new prompts) and exploitation (choosing the most well-known prompts) is crucial to Q-learning. In order to prevent the model from stagnating and to keep finding potentially better prompts that improve response quality, this trade-off is crucial.

Automated Metrics

- The quality of generated responses is quantitatively evaluated using automated evaluation metrics like **ROUGE** and **BLEU**, which offer unbiased feedback for the optimization process.

- **ROUGE - R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation
  - Measures the overlap of n-grams between generated text and reference summaries.
  - To determine the relevance and completeness of the content.

- **BLEU - Bil**ingual **E**valuation **U**nderstudy
  - Emphasizes accuracy and fluency in language generation
  - Assesses the quality of text by comparing n-grams of the produced output to reference translations.

Metrics Demonstration

Evaluation Of BLEU and ROUGE :

```
# Compute BLEU
bleu_results = self.bleu.compute(predictions=predictions, references=references)
bleu_score = bleu_results['score']  # typically 0-100

# Compute ROUGE
# Note: rouge in evaluate usually accepts references as list of strings or list of lists.
# using list of lists for consistency with BLEU here.
rouge_results = self.rouge.compute(predictions=predictions, references=references)
rougeL_score = rouge_results['rougeL']  # typically 0-1
```

Output:

```
Generated: The cat sat
Reference: The cat sat on the mat
Calculated Reward: 0.3333
```

Implementation and Training Process

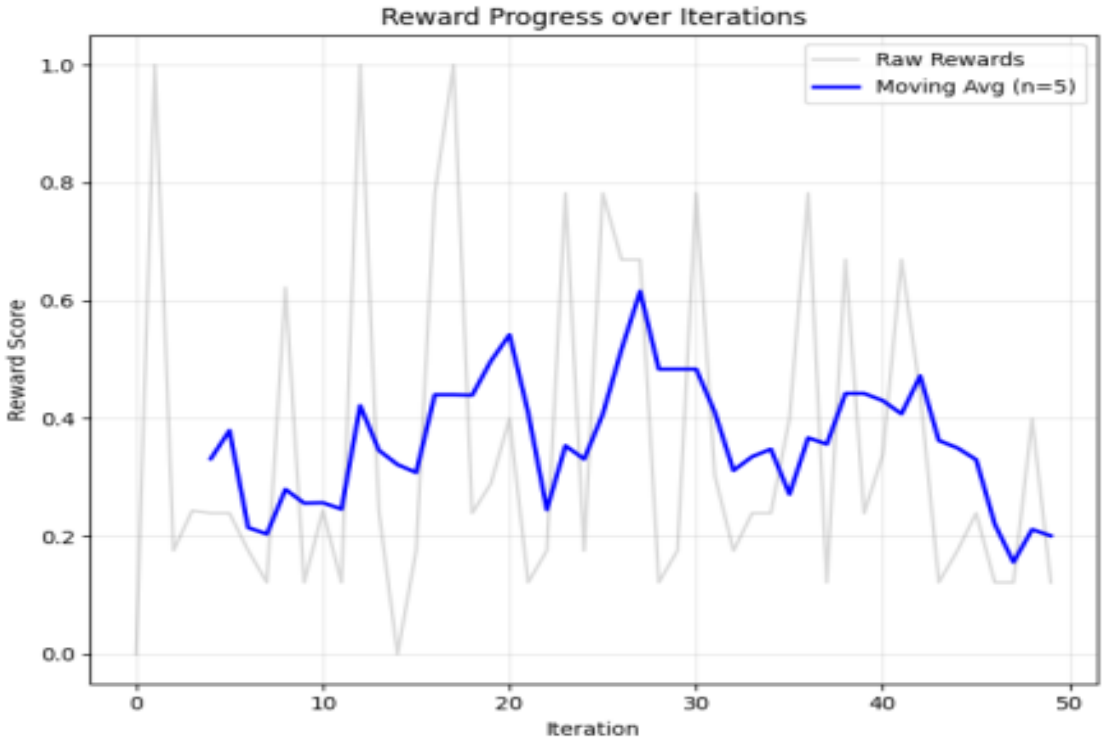
- The implementation of RLHF-Lite utilizes popular machine learning frameworks and libraries, such as Transformers and Torch to facilitate model training and optimization.
- **Training and Data loop Structure**:
  - In order to ensure continuous improvement, the training process entails an organized loop in which prompts are repeatedly tested, assessed, and improved in response to feedback from the reward engine.
- **Evaluation Metrics and Computation Rewards**:
  - The Q-learning agent's learning process is guided by reward computation, and evaluation metrics are essential for measuring the performance of generated outputs.

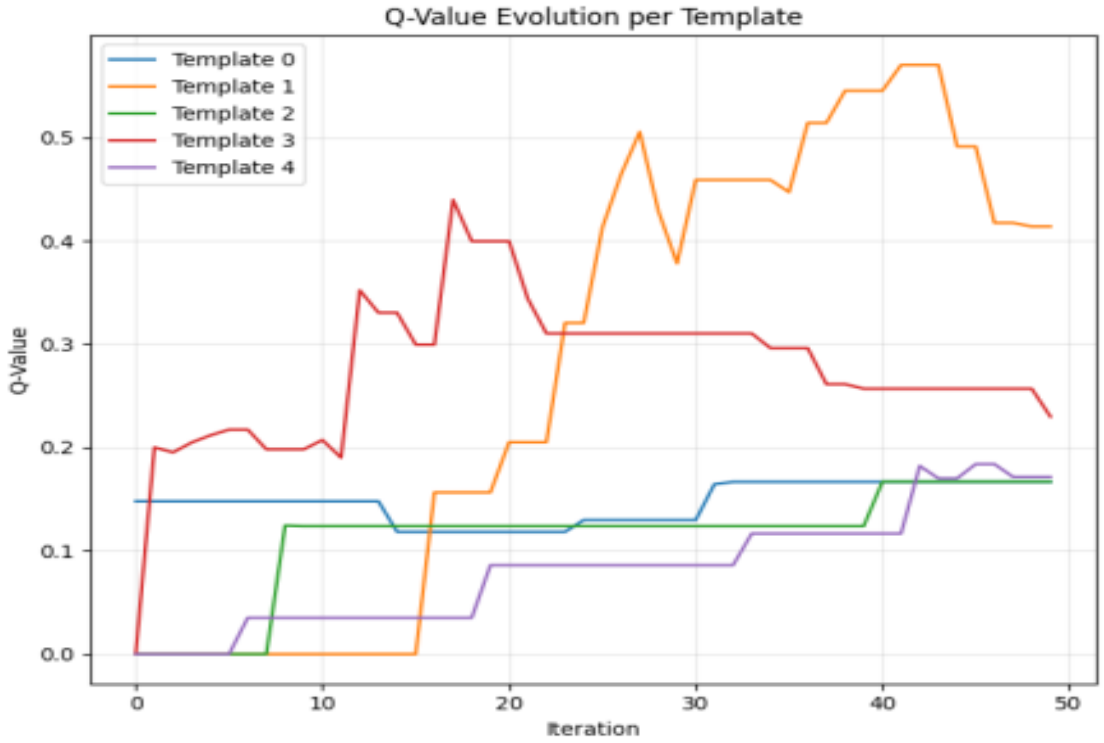
Code : Prompt Comparison

```
# BEFORE vs AFTER PROMPT COMPARISON


test_inputs = [
    "The quick brown fox jumps over the lazy dog while the sun sets behind the mountains.",
    "Artificial Intelligence is transforming industries like healthcare and finance.",
    "The spacecraft drifted silently through the empty darkness of space."
]

print("🔍 BEFORE vs AFTER Prompt Comparison")
print("="*60)

# BEFORE = Template 0
before_prompt = optimizer.templates[0]

# AFTER = Template with highest Q-value
best_idx = max(optimizer.q_values, key=optimizer.q_values.get)
after_prompt = optimizer.templates[best_idx]

for text in test_inputs:

    # append the input text
    before_full = f"{before_prompt}\n{text}"
    after_full  = f"{after_prompt}\n{text}"
```

Output : Results

- **Reward Trend:** The moving-average reward curve became more stable over time, showing that the agent gradually learned which prompts perform best.
- **Q-Value Learning:** Template “TL;DR:” consistently received the highest Q-value and became the optimal summarization prompt.





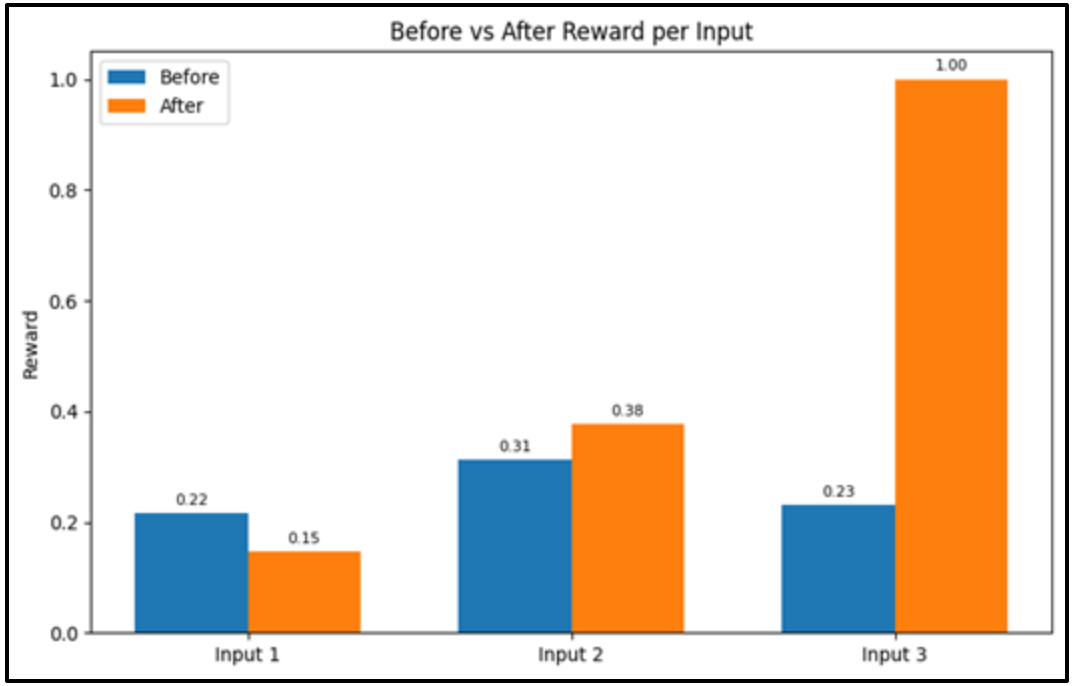
Output : Results contd...

Before vs After: Summary Quality Comparison

Across the three evaluation examples:

- **Input 3** (Spacecraft passage) showed the largest improvement.
  - Baseline reward: 0.23
  - After RL optimization: 1.0

- The optimized prompt produced a summary perfectly aligned with the reference, outperforming the baseline significantly.
- **Input 2** (AI passage) showed a moderate improvement (0.31 → 0.38).
- **Input 1** (Fox sentence) remained mostly unchanged, indicating that extremely short and simple text offers less room for improvement.



Challenges

- Understanding model & prompt behavior
- Data preparation & cleaning issues
- Reinforcement learning instability
- Computational limitations (GPU/Memory)
- BLEU & ROUGE evaluation constraints
- Debugging & integration complexity
- Hyperparameter tuning challenges
- Generalization limitations

Future Work

- Explore semantic reward models such as BERT Score and BLEURT for deeper meaning-based evaluation instead of lexical metrics like ROUGE.
- Replace Q-learning with more advanced RL methods such as PPO (used in Instruct GPT) or modern approaches like GRPO, enabling multi-step prompt refinement.
- Scale training to larger summarization datasets (e.g., XSUM, CNN/DailyMail) for better generalization.
- Incorporate real human preference feedback to move closer to full RLHF.
- Extend the RLHF-Lite framework to other tasks such as question answering, rewriting, or style transformation, using lightweight models and quantization (e.g., QLoRA).

Conclusion

- This project demonstrated that a lightweight reinforcement learning framework (RLHF-Lite) can effectively optimize prompts for Large Language Models.
- Using FLAN-T5-Base as a frozen target model and a simple Q-learning agent, the system identified prompts that consistently produced higher-quality summaries.
- Results showed that some inputs improved significantly with optimized prompting, while others changed only slightly—highlighting the prompt sensitivity of LLMs.
- The RL agent frequently converged toward the “TL;DR:” template, suggesting that certain prompts are broadly effective under ROUGE/BLEU evaluation.
- Overall, the project proves that reinforcement learning can automate prompt engineering even with small datasets and low computational cost, providing a promising foundation for future extensions in tasks beyond summarization.

References

1. T. Zhang et al., “*BERTScore: Evaluating Text Generation with BERT*,” arXiv preprint arXiv:1904.09675, 2020.
2. T. Sellam, D. Das, and A. Parikh, “*BLEURT: Learning Robust Metrics for Text Generation*,” arXiv preprint arXiv:2004.04696, 2020.
3. L. Ouyang et al., “*Training language models to follow instructions with human feedback,*” arXiv preprint arXiv:2203.02155, 2022.
4. Chung, Hyung Won, et al. "Scaling instruction-finetuned language models." Journal of Machine Learning Research 25.70 (2024): 1-53.

Thank you

Q & A