LLM week 3

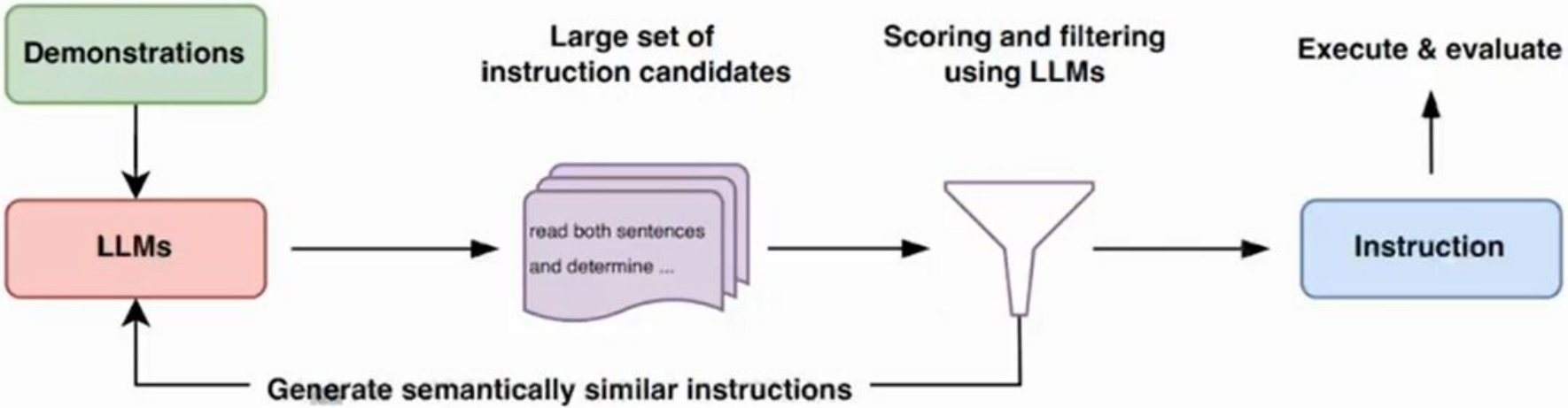
Automatic Prompt Engineering

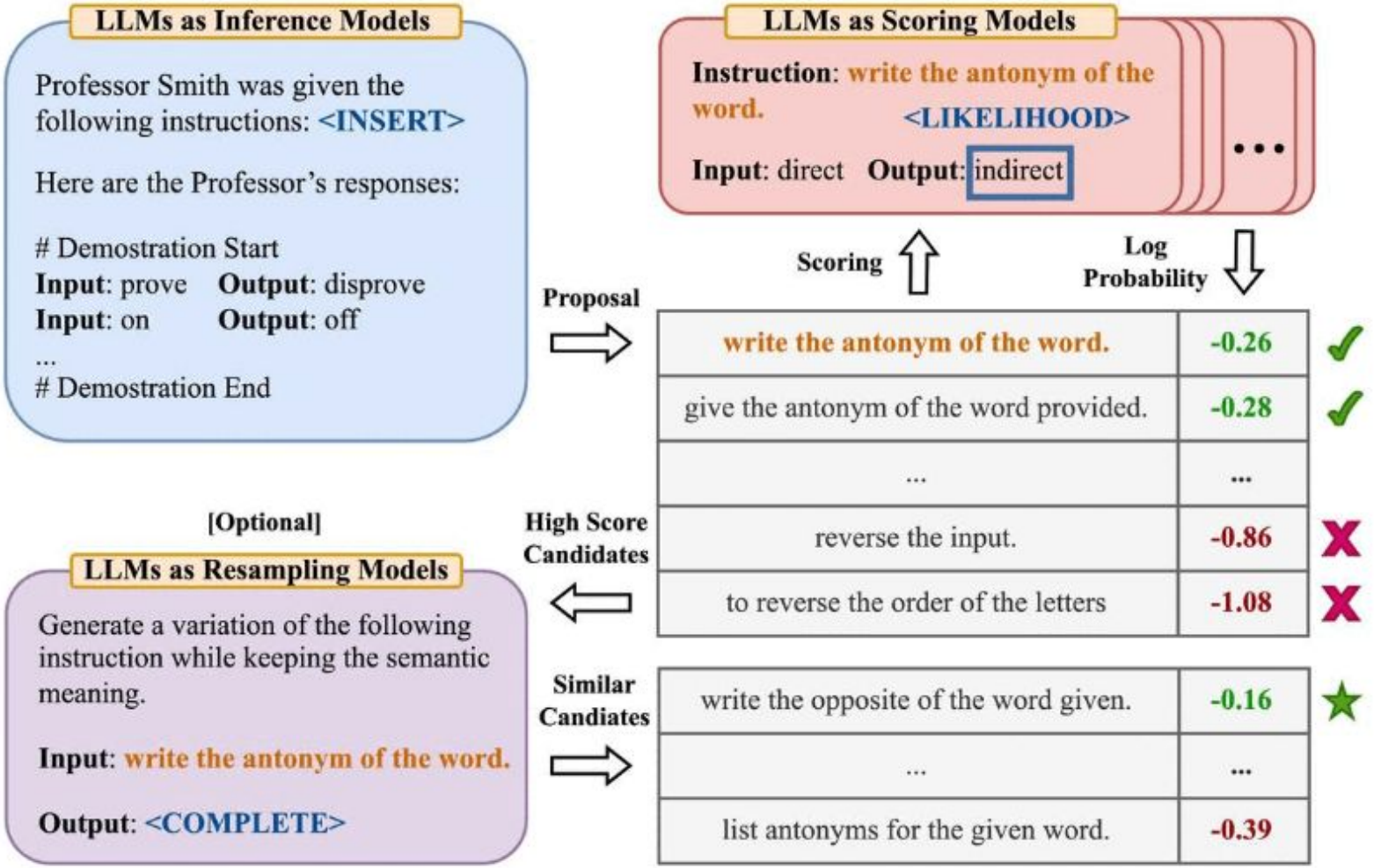
Rather than manually designing prompts for a task, automatic prompt generation leverages algorithms and frameworks to identify the most effective prompts that can drive the model to give accurate, relevant, or desired responses.

OPRO:
At the heart of the OPRO program is an algorithm called “Meta-Prompt”. It reviews previous prompts, and measures their success in solving a given problem. The algorithm then generates multiple prompts and tries them out to find the best one. In this way, Meta-Prompt can be compared to a person typing multiple variations of prompts on their keyboard until they find the most effective one.

Automatic Prompt Engineer (APE):
APE performs this in two steps:

A LLM is used to generate a set of candidate prompts.
A prompt evaluation function considers the quality of each candidate prompt; returning the prompt with the highest evaluation score. A practical implementation is, via a human-in-the-loop approach, prompts can be marked up and marked down for use on terms of accuracy and correctness.


Demonstrations
LLMs
Large set of
instruction candidates
read both sentences
and determine ...
Scoring and filtering
using LLMs
Execute & evaluate
Instruction
Generate semantically similar instructions


LLMs as Inference Models
Professor Smith was given the following instructions: <INSERT>
Here are the Professor's responses:
Demostration Start
Input: prove Output: disprove
Input: on Output: off
...
Demostration End
LLMs as Scoring Models
Instruction: write the antonym of the word.
<LIKELIHOOD>
Input: direct Output: indirect
Scoring
Log Probability
Proposal
write the antonym of the word. -0.26
give the antonym of the word provided. -0.28
... ...
reverse the input. -0.86
to reverse the order of the letters -1.08
High Score Candidates
[Optional]
LLMs as Resampling Models
Generate a variation of the following instruction while keeping the semantic meaning.
Input: write the antonym of the word.
Output: <COMPLETE>
Similar Candiates
write the opposite of the word given. -0.16
... ...
list antonyms for the given word. -0.39

Original Input $\boldsymbol{x}_{\text{inp}}$

a real joy.

Trigger Tokens $\boldsymbol{x}_{\text{trig}}$

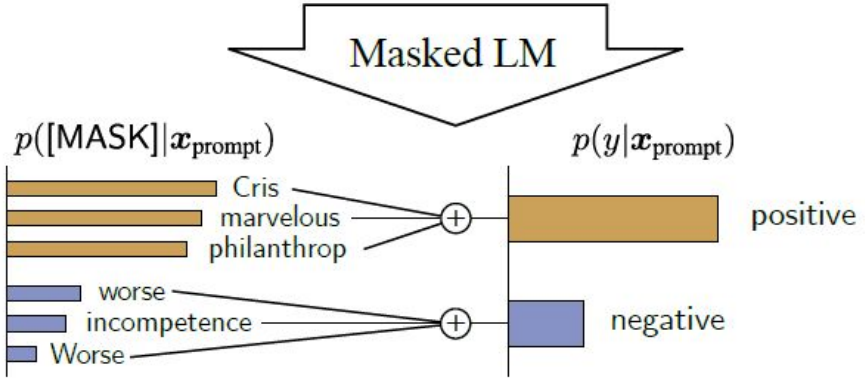
atmosphere, alot, dialogue, Clone...

Template $\lambda(\boldsymbol{x}_{\text{inp}}, \boldsymbol{x}_{\text{trig}})$

{sentence}[T][T][T][T][T][P].

AUTOPROMPT $\boldsymbol{x}_{\text{prompt}}$

a real joy. atmosphere alot dialogue Clone totally [MASK].



Task	Prompt Template	Prompt found by AUTOPROMPT	Label Tokens
Sentiment Analysis	{sentence} [T]...[T] [P].	unflinchingly bleak and desperate Writing academicswhere overseas will appear [MASK].	**pos**: partnership, extraordinary, ##bla **neg**: worse, persisted, unconstitutional
NLI	{prem}[P][T]...[T]{hyp}	Two dogs are wrestling and hugging [MASK] concretepathic workplace There is no dog wrestling and hugging	**con**: Nobody, nobody, nor **ent**: ##found, ##ways, Agency **neu**: ##ponents, ##lary, ##uated
Fact Retrieval	*X plays Y music* {sub}[T]...[T][P].	Hall Overton fireplacemade antique son alto [MASK].	
Relation Extraction	*X is a Y by profession* {sent}{sub}[T]...[T][P].	Leonard Wood (born February 4, 1942) is a former Canadian politician. Leonard Wood gymnasium brotherdicative himself another [MASK].	

Natural Language Inference performance on the SICK-E test set and variants.

Model	SICK-E Datasets		
	standard	3-way	2-way
Majority	56.7	33.3	50.0
BERT (finetuned)	86.7	84.0	95.6
BERT (linear probing)	68.0	49.5	91.9
RoBERTa (linear probing)	72.6	49.4	91.1
BERT (AUTOPROMPT)	62.3	55.4	85.7
RoBERTa (AUTOPROMPT)	65.0	69.3	87.3

Prompt Type	Original			T-REx		
	MRR	P@10	P@1	MRR	P@10	P@1
LAMA	40.27	59.49	31.10	35.79	54.29	26.38
LPAQA (Top1)	43.57	62.03	34.10	39.86	57.27	31.16
AUTOPROMPT 5 Tokens	53.06	72.17	42.94	54.42	70.80	45.40
AUTOPROMPT 7 Tokens	53.89	73.93	43.34	54.89	72.02	45.57

Model	MRR	P@10	P@1
BERT	55.22	74.01	45.23
RoBERTa	49.90	68.34	40.01

Table 4: **Factual Retrieval:** On the left, we evaluate BERT on fact retrieval using the *Original* LAMA dataset from Petroni et al. (2019). For all three metrics (mean reciprocal rank, mean precision-at-10 (P@10), and mean precision-at-1(P@1)), AUTOPROMPT significantly outperforms past prompting methods. We also report results on a *T-REx* version of the data (see text for details). On the right, we compare BERT versus RoBERTa on a subset of the LAMA data using AUTOPROMPT with 5 tokens.

CoreCORE: Resolving Code ality Issues using LLMs

Python code flagged by the quality check

```python
def separate_headers(files):
    for file in files:
        lines = iter(file)
        header = next(lines)
        body = [l for l in lines]
        yield header, body
```

Fix recommendation

Each call to `next()` should be wrapped in a `try-except` block to explicitly handle `StopIteration` exceptions.

Revised Python code fixing the issue

```python
def separate_headers(files):
    for file in files:
        lines = iter(file)
        try:
            header = next(lines)
        except StopIteration:
            continue
        body = [l for l in lines]
        yield header, body
```

Java code flagged by the quality check

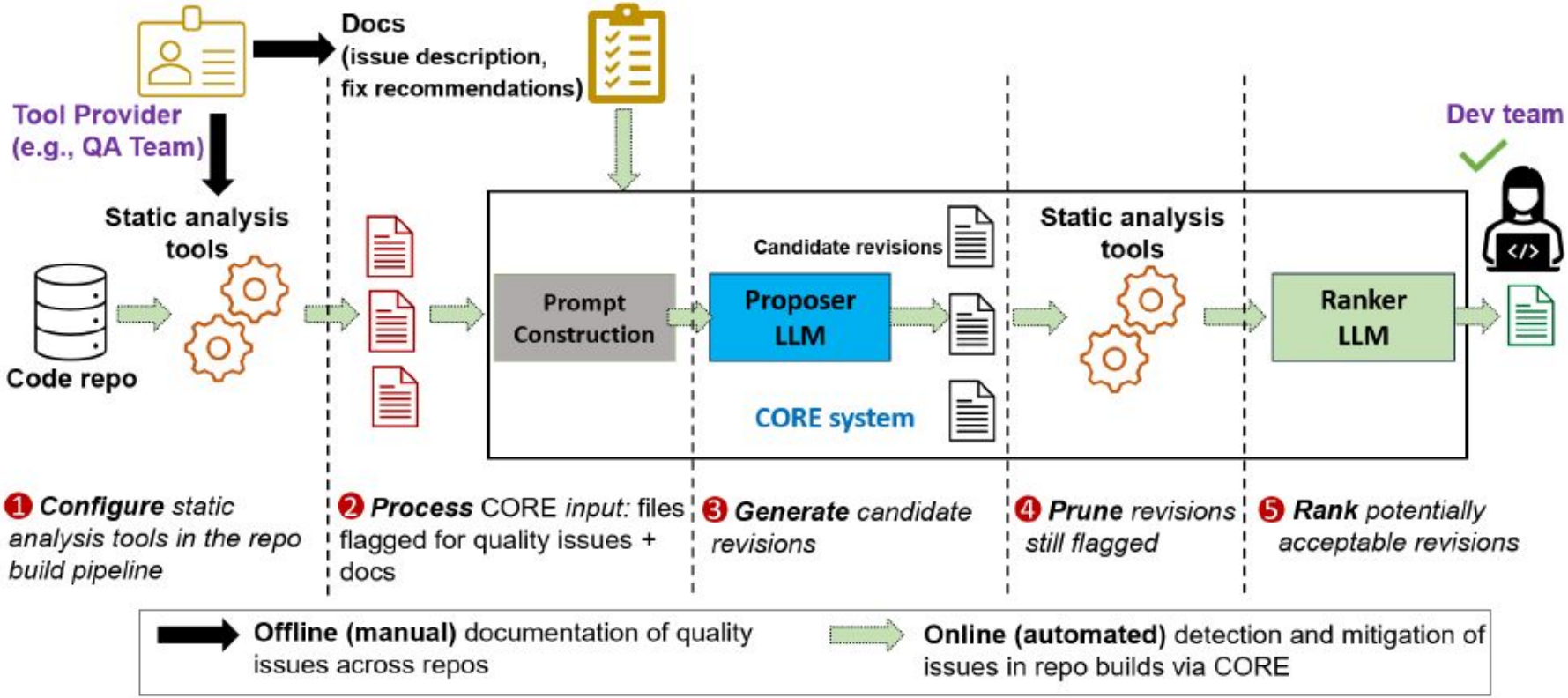
```java
class ComputePrimesThread extends Thread {
    @Override
    public void run() {
        // ...
    }
}
new ComputePrimesThread().run();
```

Fix recommendation

If you intend to execute the contents of the `Thread.run()` method with a new thread, call `Thread.start()` instead.

Revised Java code fixing the issue

```java
class ComputePrimesThread extends Thread {
    @Override
    public void run() {
        // ...
    }
}
new ComputePrimesThread().start();
```


Docs
(issue description,
fix recommendations)
Tool Provider
(e.g., QA Team)
Static analysis
tools
Code repo
Prompt
Construction
Proposer
LLM
Candidate revisions
CORE system
Static analysis
tools
Ranker
LLM
Dev team
❶ Configure static analysis tools in the repo build pipeline
❷ Process CORE input: files flagged for quality issues + docs
❸ Generate candidate revisions
❹ Prune revisions still flagged
❺ Rank potentially acceptable revisions
Offline (manual) documentation of quality issues across repos
Online (automated) detection and mitigation of issues in repo builds via CORE

Proposer Prompt Template

p1 Description of the quality issue.
p2 Recommendations for resolving the quality issue.
p3 (Optional) Relevant code blocks for doing the revision.
p4 Input source file (in full, or localized to the block containing the issue).
p5 Location and warning message given by the static check.

Ranker Prompt Template

r1 Description of the quality issue. *(same as* p1 *in the Proposer template.)*
r2 Recommendations for resolving the quality issue. *(same as* p2 *in the Proposer template.)*
r3 Rubric for scoring the revisions on the scale of "Strong Reject", "Weak Reject", "Weak Accept", "Strong Accept".
r4 Input candidate revision as "Diff" with its source file.

Dataset	Dataset statistics		Effectiveness of Proposer LLM		Rankings by Ranker LLM	
	#Files flagged	#Issues flagged (Avg. per file)	#Files passing static checks (%)	#Issues remaining (Avg. per file)	#Files ranked high (%)	#Files ranked low (%)
CQPy	2752 (100%)	5389 (1.95)	2444 (88.81%)	693 (0.25)	2325 (84.48%)	119 (4.32%)
CQPyUS	520 (100%)	999 (1.90)	453 (87.11%)	159 (0.31)	427 (82.11%)	26 (5.00%)
SQJava	483 (100%)	999 (2.06)	397 (82.19%)	270 (0.56)	371 (76.81%)	26 (5.38%)

Summary of end-to-end evaluation of CORE on real-world Python and Java files, with 52 and 10 static checks using CodeQL and SonarQube respectively.

Stage evaluated	Stage-wise output		Results of user study		
	#Files retained	#Revisions retained	% Files accepted (#)	% Revisions accepted (#)	% Revisions rejected (#)
Stage 4 (Proposer LLM)	453 (100%)	2397 (100%)	70.64% (320)	44.89% (1076)	55.11% (1321)
Stage 5 (Ranker LLM, SA)	410 (100%)	1756 (100%)	**72.68%** (298)	**52.45%** (921)	47.55% (835)
Stage 5 (Ranker LLM, WA)	17 (100%)	228 (100%)	58.82% (10)	36.40% (83)	63.60% (145)
Stage 5 (Ranker LLM, WR/SR)	26 (100%)	413 (100%)	46.15% (12)	17.43% (72)	**82.57%** (341)

Comparison of CORE with the state-of-the-art APR technique Sorald on the SQJava dataset consisting of 10 static checks, using SonarQube as the static analysis tool.

	#Files (%)	#Issues remaining (%)
Flagged	483 (100%)	999 (100%)
CORE	371 (76.8%)	270 (27.03%)
Sorald	378 (78.3%)	371 (37.14%)

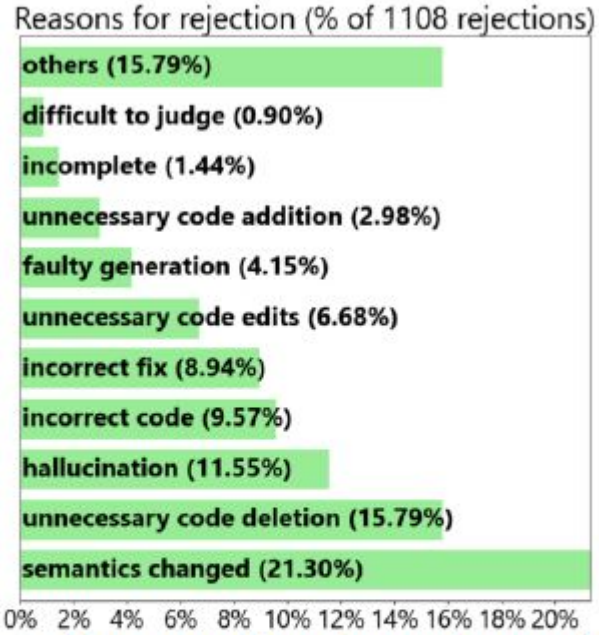


Fig. 5. Reasons given by reviewers for rejecting the candidate revisions produced by CORE on CQPyUS.

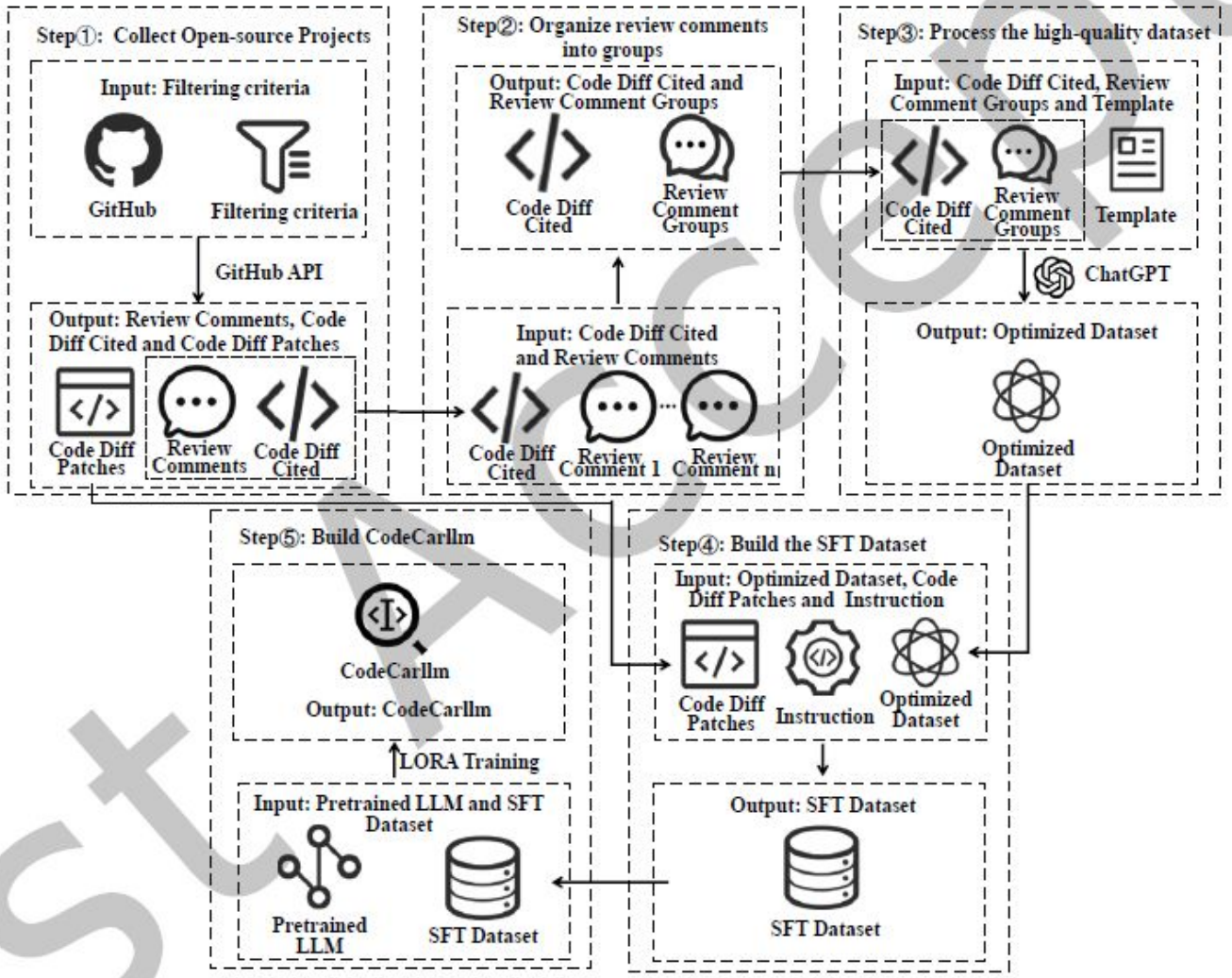
Fine-tuning Large Language Models to Improve Accuracy and Comprehensibility of Automated Code Review

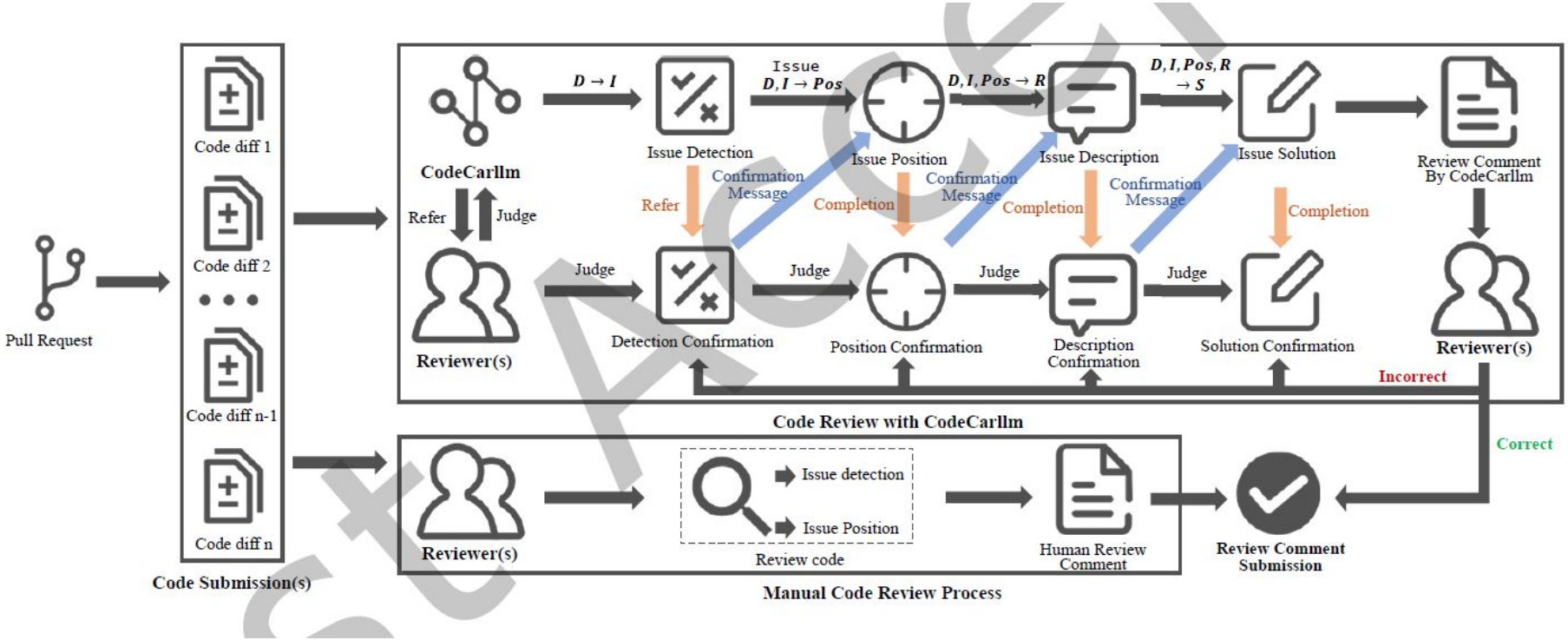
ACR

(a) accurately detect and localize the issue,

(b) provide a reasonable explanation for the cause of the issue

(c) suggest a concrete repair solution to fix the issue within a diff


Step①: Collect Open-source Projects
Input: Filtering criteria
GitHub
Filtering criteria
GitHub API
Output: Review Comments, Code Diff Cited and Code Diff Patches
Code Diff Patches
Review Comments
Code Diff Cited
Step②: Organize review comments into groups
Output: Code Diff Cited and Review Comment Groups
Code Diff Cited
Review Comment Groups
Input: Code Diff Cited and Review Comments
Code Diff Cited
Review Comment 1
Review Comment n
Step③: Process the high-quality dataset
Input: Code Diff Cited, Review Comment Groups and Template
Code Diff Cited
Review Comment Groups
Template
ChatGPT
Output: Optimized Dataset
Optimized Dataset
Step⑤: Build CodeCarllm
CodeCarllm
Output: CodeCarllm
LORA Training
Input: Pretrained LLM and SFT Dataset
Pretrained LLM
SFT Dataset
Step④: Build the SFT Dataset
Input: Optimized Dataset, Code Diff Patches and Instruction
Code Diff Patches
Instruction
Optimized Dataset
Output: SFT Dataset
SFT Dataset


Pull Request
Code diff 1
Code diff 2
Code diff n-1
Code diff n
Code Submission(s)
CodeCarllm
Refer
Judge
Reviewer(s)
D → I
Issue Detection
Issue
D, I → Pos
Issue Position
D, I, Pos → R
Issue Description
D, I, Pos, R → S
Issue Solution
Review Comment By CodeCarllm
Refer
Confirmation Message
Completion
Confirmation Message
Completion
Confirmation Message
Completion
Judge
Detection Confirmation
Judge
Position Confirmation
Judge
Description Confirmation
Judge
Solution Confirmation
Reviewer(s)
Incorrect
Correct
Code Review with CodeCarllm
Reviewer(s)
Issue detection
Issue Position
Review code
Human Review Comment
Review Comment Submission
Manual Code Review Process

Your task is to translate the information received from the review comments into a standard review comment. You should return a json string. json contains the keys **'hasIssue'** and **'ReviewComments'**. **'ReviewComments'** are specific to the code problem and contain a **'IssuePosition'** indicating the location of the problem, a **'IssueDescription'** explaining the problem in detail, and a **'IssueSolution'** of possible solutions.
An example is shown below.

code diff:

```
define(function(require) {
        * @return {void}
        */
       function updateHandlers(dateExtent) {
-          if (dateExtent == null) {
+          if (dateExtent === null) {
```

conversations:

nan: Comparing against `null` with the `===` operator only checks that the value is `null`, but not `undefined`. If you would like to check for both `undefined` and `null`, you can simply use `==`. This will coerce them to the same value, which means that you just need to check against `null` for both. Nothing else will ever be coerced to `null` or `undefined`, so this is the one occasion where `==` is safe to use.
If this is the case, I would suggest making this change:

```javascript
        if (dateExtent == null) {
```

Golodhros: Thanks, this is good for now.

standard review comments:

```
{
   \"hasIssue\":true,
   \"ReviewComments\":[
      {
         \"IssuePosition\":\"- if (dateExtent == null) {\\n+ if (dateExtent === null) {\",
         \"IssueDescription\":\"The code uses the strict equality operator (`===`) to compare `dateExtent` with `null`. This comparison will return `true` only if `dateExtent` is `null` and of the same type (`null`). If `dateExtent` is `undefined`, `false`, or any other value, the condition will evaluate to `false`.\",
         \"IssueSolution\":\"Depending on the intended behavior, it might be necessary to use the loose equality operator (`==`) instead.\"
      }
   ]
}
```

Process the following data according to the instructions and example.

code diff:

```
{}
```

conversations:

```
{}
```

standard review comments:

Table 5. Experimental settings

RQ	ACR Method	Evaluation Steps	Datasets
RQ1	***MagicCarllm*** ***Carllm*** ***Carllm2*** ***CodeCarllm*** CodeBert CodeT5 CodeReviewer LLaMA-Reviewer Magicoder* LLaMA1* LLaMA2* CodeLLaMA* ChatGPT-0613* GPT-4-0613*	Step 1. Use the same dataset for training. Step 2. Detect issues from diffs in the testing dataset. Step3. Evaluate performance using Recall, Precision, F1-score and Accuracy.	A same **training dataset** contains 19,456 instances (diff-review comment pairs, both positive and negative, see subsection 3.2.3) prepared for *Carllm*, CodeBert, CodeT5, CodeReviewer and LLaMA-Reviewer. The **testing dataset** contains 1000 positive instances and 1000 negative instances which have not been included in the training dataset.
RQ2	***MagicCarllm*** ***Carllm*** ***Carllm2*** ***CodeCarllm*** CodeReviewer CodeT5 LLaMA-Reviewer	Perform manual evaluation to determine the comprehensibility of the review comments generated by various methods. Specifically, we use 'degree of clarity' to measure comprehensibility, meaning that to what degree a review comment clearly explains the issue in a code diff.	The **testing dataset** contains 1000 positive instances .

No training is needed for these approaches.

Table 6. Performance of each method under the code issue detection task

Model&Method	Recall	Precision	F1-Score	Accuracy
CodeBert	0.5830	0.7049	0.6382	0.6695
CodeT5	0.6180	0.6828	0.6488	0.6655
CodeReviewer	0.6050	0.7084	0.6526	0.6780
LLaMA-Reviewer	0.7850	0.6066	0.6844	0.6380
Magicoder 7B (2-shot)	0.5202	0.7460	0.6130	0.5290
LLaMA 13B (2-shot)	0.5520	0.4855	0.5166	0.4835
LLaMA2 13B (2-shot)	0.3380	0.5113	0.4070	0.5075
CodeLLaMA 13B (2-shot)	0.5960	0.5326	0.5625	0.5365
ChatGPT-0613 (2-shot)	0.1990	0.5113	0.2851	0.5010
GPT-4-0613 (2-shot)	0.1850	0.4684	0.2652	0.4875
MagicCarllm 7B	0.5790	**0.7500**	0.6535	0.6930
Carllm 13B	**0.8500**	0.6268	0.7215	0.6720
Carllm2 13B	0.6950	0.6874	0.6912	0.6895
CodeCarllm 13B	0.7700	0.6968	**0.7316**	**0.7175**

Dynamic (Contextual) Prompting

involves providing a context in the prompt to guide the model's response.

This technique helps the model understand the context and perform the task better.

```
context = "You are a customer support agent. A customer is unhappy with their recent purchase."
prompt = f"{context} Write a polite response to address their concerns."
```

The challenge here is, how to format the context.

Challenges

- **Context Overflow (Token Limit):** LLMs have token limits, supplying large contexts may exceed that limit.

  Solution: Recent context can be prioritized or the context can be summarized.

- **Context Relevance:** Including irrelevant information confuses the model.

  Solution: Ranking algorithms like cosine similarity, TF-IDF, BM25, etc., to ensure only relevant context is used.

- **Context Drift:** Over time, old context may become irrelevant.

  Solution: Regularly refresh, or summarize the context to keep it aligned with the current conversation.

```python
def get_response(query, context):
    # Create the context prompt by joining the previous interactions
    context_prompt = "\n".join(context)
    # Prepare the full prompt with context
    prompt_template = (
        f"Conversation History:\n\n{context_prompt}.\n\n Please answer the following question:\n\n{query}"
    )
    # Get response from the model
    if context_prompt == "":
        response = model.invoke(query)
        print("Prompt: ", query)
    else:
        response = model.invoke(prompt_template)
        print("Prompt: ", prompt_template)
    # Update context
    context.append(f"User: {query}")
    context.append(f"AI: {response}")
    return response
```

[12] ✓ 0.0s Python

```python
response1 = get_response("What's the capital of France?", context)
print("Answer:",parser.invoke(response1))
response2 = get_response("What is the population of that city?", context)
print("Answer:",parser.invoke(response2))
```

[13] ✓ 1.9s Python

```
Prompt:  What's the capital of France?
Answer: The capital of France is **Paris**.

Prompt:  Conversation History:

User: What's the capital of France?
AI: content='The capital of France is **Paris**. \n' additional_kwargs={} response_metadata={'is_blocked': False,

 Please answer the following question:

What is the population of that city?
Answer: The population of Paris is approximately **2.14 million**.

Keep in mind that this is just the population of the city of Paris itself, not the entire Paris metropolitan area
```

Frameworks, Tools

There are countless Plug-and-play high level chatbots etc.

We have options like **LangChain and Hugging Face**, both have their own pros and cons for our purposes in this project.

Hugging Face focuses on offering wide range of pretrained models. It is desirable in scenarios involve general NLP tasks.

LangChain is a rich framework to build LLM applications that interacts with external tools

Hugging Face **doesn't offer a native context/history management.** Developer needs to implement manually. So in scenarios with quick, simple tasks like classification and text summarization, Hugging Face can be a good option since it offers many pretrained models.

LangChain **offers memory management** (We used in week 1 project already) to keep the history automatically, and integration with external tools such as model APIs. It is generally desirable for scalable, multi-step workflows, **so LangChain would be a better decision for projects like ours.**

Retrieval Augmented Prompting (RAP)

This technique resembles the Contextual Prompting in terms of enhancing the prompt by prepending some other content.

But RAP differs by the content it puts in prompt. Here we retrieve relevant content from a **knowledge base** not from the conversation history or prompt template.

This helps improve the accuracy and relevance of responses by providing **real-time and domain-specific** information to the model.

The challenge: Which and how many documents to fetch?

Challenges

Context Relevance: Retrieving irrelevant documents or information can confuse the model and reduce response quality.

- **Solution**: Use **ranking algorithms** or **semantic search** to ensure only the most relevant context is retrieved.
- **Example**: Ranking documents based on cosine similarity to the query using a vector database (e.g., FAISS).

Token Limitations: Prompting models with too much retrieved information may exceed token limit.

- **Solution**: Retrieve and select only the top-K most relevant documents or summarize the retrieved data before adding to the prompt.
- **Example**: Retrieving the top 3 documents from a vector store and including their summaries instead of full texts.

Efficiency and Latency: Retrieving data from external sources can slow down response time.

- **Solution**: Use **caching**, pre-fetching, or optimizing retrieval algorithms for speed.

```python
# Sample documents retrieved from a vector database
retrieved_docs = [
    {"content": "Google Cloud provides a suite of cloud computing services."},
    {"content": "Vertex AI is a managed machine learning platform on Google Cloud."}
]

# Example query
query = "How does Google Cloud support machine learning?"

# Combine retrieved documents into a context
context = " ".join([doc["content"] for doc in retrieved_docs])

# Prepare the full prompt
prompt = f"Here is some information about Google Cloud: {context}\n\nUser Query: {query}\nAnswer:"
print("Prompt: ",prompt)
```

Python

```
Prompt:  Here is some information about Google Cloud: Google Cloud provides a suite of cloud computing services. Vertex AI is a manage

User Query: How does Google Cloud support machine learning?
```

```python
# Generate the response
response = model.invoke(prompt)

print(parser.invoke(response))
```

Python

```
Google Cloud offers a robust and comprehensive suite of services to support machine learning, with Vertex AI being a key componen

**Vertex AI: The Core of Machine Learning on Google Cloud**

* **Managed Machine Learning Platform:** Vertex AI acts as a central hub for all your machine learning needs, offering tools for:
    * **Data Preparation:** Cleaning, labeling, and organizing your data for optimal model training.
    * **Model Training:** Building and training models using various algorithms, pre-trained models, and customizable configurati
    * **Model Deployment and Management:** Serving your trained models for predictions and managing their lifecycle, including mo
    * **Experiment Tracking and Optimization:** Tracking experiments, comparing model performance, and optimizing hyperparameters
    * **AI Explainability:** Understanding how models make predictions, promoting transparency and trust.

**Beyond Vertex AI: A Comprehensive Ecosystem**

* **BigQuery ML:** A powerful tool for building and running machine learning models directly within BigQuery, Google's data wareh
* **AI Platform Pipelines:** For streamlining and automating your machine learning workflows, from data preparation to model depl
* **Pre-trained Models:** Access a vast library of pre-trained models covering various domains like natural language processing,
* **Compute Engine:**  Use virtual machines with powerful GPUs and TPUs for demanding model training and inference tasks.
* **Kubernetes Engine:** Deploy and manage your machine learning applications in a containerized environment for scalability and
* **Cloud Storage:** Store your data securely and efficiently on Google's cloud infrastructure.
* **Cloud Vision API, Natural Language API, and more:** These APIs provide pre-built machine learning capabilities for specific t

**Key Benefits of Google Cloud for Machine Learning:**

* **Scalability:** Easily scale your machine learning resources up or down as needed.
* **Cost-effectiveness:** Pay only for the resources you use, with flexible pricing options.
* **Innovation:** Access the latest technologies and research from Google AI.
* **Security and Privacy:** Benefit from Google's robust security measures and compliance certifications.
* **Integrations:** Seamlessly integrate with other Google Cloud services for a unified experience.
```

Frameworks, Tools

Again we see LangChain and Hugging Face as tools for this type of prompting, we already discussed the general characteristics.

LangChain again looks better for tasks like ours in this project, especially for one important aspect: retrieving **relevant** documents **fast.** There are some tools to achieve this without needing manual optimization of this retrieval. We can use such tool integrations with LangChain.

Pinecone:

- A **vector database** that stores documents in a way that allows for fast, semantic retrieval.
- Commonly used with Langchain to fetch the most relevant context to improve model responses.

FAISS:

- A popular **open-source library** for **similarity search** and **clustering** of dense vectors.
- Efficient for **embedding-based retrieval**, often used to fetch documents based on their similarity to the query.

Feature	Pinecone	FAISS
Type	Managed vector database	Open-source similarity search library
Scalability	Handles billions of vectors easily	Efficient for large datasets but requires more setup
Ease of Use	Simple API with real-time capabilities	Requires more configuration and coding
Updates	Real-time updates with dynamic indexing	Updates require re-indexing
Performance	Optimized for low-latency queries	High-performance but may vary based on index type
Cost	Pay-as-you-go pricing	Free, but self-managed infrastructure costs

Database Research

While looking for a dataset, I got sure it is going to be hard.

We have some general code datasets, and maybe we might look and scrape some repos with spark code, some kaggle competitions that might require Spark.

We also have a couple of blog posts, or tutorials about the best practices so we can either write our own code accordingly or somehow train the model with them.

https://developer.ibm.com/blogs/spark-performance-optimization-guidelines/
https://www.reddit.com/r/datasets/comments/13vs2eg/list_of_code_generation_datasets_open_source/
https://www.kaggle.com/datasets/veeralakrishna/150k-python-dataset
https://spark.apache.org/docs/latest/api/python/user_guide/pandas_on_spark/best_practices.html
https://medium.com/@ashwin_kumar_/spark-performance-tuning-best-practices-67d564aafa03
https://sparkbyexamples.com/spark/spark-performance-tuning/
https://umbertogriffo.gitbook.io/apache-spark-best-practices-and-tuning