Open Ended Questions:

Try the finetuning process on Gemma-2b.

Check why phi-3 is mimicking the dataset more than others.

Comparison of Codegen2-7B and Mistral 7b for Code Review Tasks

#	Model	Codegen2-7B_P	Mistral 7b Instruct
1.	returnns a	`def returnns_a():` `return a`	1. Spelling mistake in `returnns` which should be `returns`. 2. Missing semicolon at the end of the line. It should be `return a;`. 3. Use lowercase 'r' for return type if it is not a reference type, e.g., `int returns functionName() {...}`. If it is a reference type, use uppercase 'R', e.g., `int R& returnsFunctionName() {...}`.
2.	result = (4 + 2 * 3))	`` ```python `` `result = (4 + 2 * 3)` `` ``` `` `### Expected Result:` `` ```python `` `result = 7`	The expression in the parentheses should be ordered according to the BIDMAS (or PEMDAS) rule: Multiplication and division before addition and subtraction. ``Correct this line: `result = (4 + 2) * 3` or `result = (2 * 4 + 3)` depending on the desired outcome.``
3.	if x=5:	`` ```python `` `if x==5:` `print("x is 5")` `else:` `print("x is not 5")` `` ``` ``	This code snippet contains a syntax error. In Python, you should use "==" instead of "=" in the if statement. The correct format is: `if x == 5:`
4.	if x == 0	`if x == 0:` `print("x is zero")`	This code snippet appears to lack necessary context and could potentially be a syntax error or an incomplete condition. Here are two potential issues: 1. The variable 'x' has not been defined before the if statement, causing a NameError. Ensure it is properly initialized. 2. The condition is missing an else or elif clause, making the code incomplete and potentially leading to unexpected behavior. `if x is None or x == 0:  # Properly check for both None and equal to zero` `# Your code here` `else:` `# Other logic here`
5.	ages = { 'pam': 24, 'jim': 24 'michael': 43 }	`ages = {'pam': 24,    'jim': 24 'michael': 43}`	The dictionary `ages` has a syntax error. Keys in dictionaries should be enclosed in quotes and a comma should separate each key-value pair. Here's the corrected version:

			`ages = {'pam': 24, 'jim': 24, 'michael': 43}`
6.	len('hello') = 5	`len('hello') = 5`	You cannot assign a value to the length of a string in Python. Use `len(str)` instead, where `str` is your string variable.

7.	fro i in range(10):	`for i in range(10):`
		`            print(i)`
			It looks like you've started a for loop without providing a variable name for the iteration. Here are two issues:
			1. Use `for i in range(10)` instead of just `for i`.
			2. Ensure you use the correct variable name `i` within your code block, as it is currently being used for the loop variable.

8.	for i range(10):	`for i in range(10):`
		`  print(i)`
			It seems you've missed specifying the data type for variable 'i' in your 'for' loop. In Go, it should be 'int'. Also, consider using 'for i := 0; i < 10; i++' instead of 'range(10)'. Here's a corrected example:
			``
			```
			go ``
			`for i := 0; i < 10; i++ {`
			`    // Your code here`
			`}`
			``
			```
			 ``

9.	ages = {
	'pam': 24,
	'jim': 24,
	'michael': 43
	}
	print(f'Michael is {ages["michael]} years old.')
		Nothing	1. Variable name `ages` is a dictionary with string keys and integer values, consider using a list of tuples instead for simplicity and better readability when working with key-value pairs.
			2. Missing a comma before the last key-value pair in the dictionary declaration.
			`ages = {`
			`    'pam': 24,`
			`    'jim': 24,`
			`    'michael': 43`
			`}`
			`print(f'Michael is {ages["michael"]}`
			`years old.')`

10.	def foo():
	return [1, 2, 3
	
	print(foo())
		`def foo():`
		`  return [1, 2, 3]`
		`print(foo())`
			Indentation is incorrect in the function definition `def foo():    return [1, 2, 3 print(foo())`. It should be `def foo():  return [1, 2, 3] print(foo())` or `def foo():  return [1, 2, 3]` for separate prints.

11.	ages = {'pam'=24}	`ages = {'pam'=24}`	This code creates a dictionary with a single key-value pair and incorrectly assigns the value to the variable named 'ages'. To fix this, use curly braces to define a dictionary properly: `ages = {'pam': 24}`

We tested both the Codegen2-7b model and Mistral 7b model with the same prompts and inputs. Now, if we look at the results, we can clearly see that Mistral 7b outperforms Codegen2-7b model.

In evaluating the Codegen2-7B and Mistral 7b models for code review, Mistral 7b clearly outperforms Codegen2-7B.

- **Codegen2-7B's Limitations:** Codegen2-7B appears primarily focused on code generation rather than analysis. It struggles to identify errors, offer corrections, or provide meaningful feedback on code quality. In many cases, it simply returns the original input or no output at all.
- **Mistral 7b's Strengths:** Mistral 7b demonstrates a stronger understanding of code structure and potential issues. It consistently provides relevant code reviews, pinpointing syntax errors, logic problems, and potential optimizations. Crucially, it offers concrete suggestions for improvement, which is also something we want in our code reviews.

Why not CodeGen2.5:

I tried using CodeGen2.5 and followed all the specified pre-requisites. However, when I ran the simple code example provided on their page, it didn't work. I encountered the following error:

```
AttributeError: 'CodeGen25Tokenizer' object has no attribute 'encoder'
```

List Down the Resources Required for the Deployment of a Language Model on AWS EC2

Model	Instance	VRAM- FP16	Disk Space	GPU	Using	Link
Mistral 7B	g5.xlarge	14GB		NVIDIA A10	VLLM	Deploy LLaMA 3, Mistral, and Mixtral, on AWS EC2 with vLLM
Mixtral 8x7B	g5.48xlarge	140GB	300GB	NVIDIA A10	VLLM	Deploy LLaMA 3, Mistral, and Mixtral, on AWS EC2 with vLLM
LLaMA 3 70B	g5.48xlarge	140GB	~~300GB~~ 150GB	NVIDIA A10	VLLM	Deploy LLaMA 3, Mistral, and Mixtral, on AWS EC2 with vLLM
LLama 70b	p4d(Reason: long request can timeout due to the 60s request timeout limit for SageMaker)					Deploy Llama 2 7B/13B/70B on Amazon SageMaker
LLaMA 3 8B	g5.xlarge	20GB	16GB	NVIDIA A10	VLLM	How to Install and Deploy LLaMA 3 Into Production?
Llama 7b, 13b, 70B	g4.dn2xlarge	16GB RAM	512GB	NVIDIA Tesla T4		How to Llama 2 | How to Deploy an Enterprise AI LLM on private cloud instance
Llama 7b	g5.4xlarge	16GB RAM			Skypilot	Serving LLM 24x Faster On the Cloud with vLLM and SkyPilot
Llama 7b	**g5.4xlarge**	16GB RAM			TGI	Deploy Llama2 with TGI on AWS EC2 | Runhouse
Llama 13b	g4dn.xlarge			NVIDIA A100	Skypilot	Serving LLM 24x Faster On the Cloud with vLLM and SkyPilot
Code LLama 34b	g5g.16xlarge		128GB	**NVIDIA T4G Tensor Core GPU**		AWS Marketplace: Code Llama 34B Instruct v1.0.0: An Advanced AI Tool for Coding Compute – Amazon EC2 Instance Types – AWS
Llama 2 13b (Sagemaker)	g5.12xlarge		96GB	NVIDIA A10G		Deploy Llama 2 7B/13B/70B on Amazon SageMaker

mixtral 8x22b	ml.p4d.24xlarge and ml.p4de.24xlarge		320GB	NVIDIA A100 Tensor Core GPUs,		Compute – Amazon EC2 Instance Types – AWS Mixtral 8x22B is now available in Amazon SageMaker JumpStart | Amazon Web Services
LLama 13b	`g5.12xlarge`		96gb	NVIDIA A10G		sagemaker-huggingface-llama-2-samples/inference/sagemaker-notebook.ipynb at master · philschmid/sagemaker-huggingface-llama-2-samples

Control RAG Output

Prompt Techniques

How prompts can change the response?

Prompt 1:

prompt_template = """Use the following pieces of context to answer the question at the end. If you don't know the answer, please think rationally answer from your own knowledge base

Unable to answer anything outside the context. Can only produce answers that exist in the provided documents.

Prompt 2:

prompt_template = """If the context is not relevant, please answer the question by using your own knowledge about the topic

Able to answer anything even outside the context. Can produce answers that exist within the provided documents or are outside of the documents.

Advanced Prompt Engineering for Reducing Hallucination

Mastering RAG: LLM Prompting Techniques For Reducing Hallucinations - Galileo

Types of Prompting Techniques:

· Thread of Thought

· Chain of Note

· Chain of Verification

· EmotionPrompt

· ExpertPrompting

· ReAct Prompting

· Chain of Knowledge

Input/ Output Filtering

1. Filter the input. This could be anything from a simple keyword filter to a machine-learning model trained to classify questions as "good" or "bad". The idea is that out-of-domain questions never reach the LLM.

1. Filter the output. Inspect the answer produced by the LLM, and reject it if it appears to be out-of-domain. For example, if you want the LLM to answer questions about legal precedents, you can reject any answer that doesn't contain a valid case citation. This has an advantage over the input filter in that you don't need to worry about the LLM producing an out-of-domain answer to an in-domain question.

GuardRails

LLMWARE

GitHub - llmware-ai/llmware: Unified framework for building enterprise RAG pipelines with small, specialized models

Ensures output by using post-processing guardrails

LLAMA Guard

Only Supports Input and Output Moderations.

NeMo GuardRail

Supports Input moderation, output moderation, topical moderation, rag retrieved chunks moderation, calling execution tools.

- Input rails: These are applied to user inputs. An input rail can either reject the input, halt further processing, or modify the input (for instance, by concealing sensitive information or rewording).
- Dialog rails: These affect the prompts given to the LLM. Dialog rails work with messages in their canonical forms and decide whether to execute an action, summon the LLM for the next step or a reply, or opt for a predefined answer.
- Execution rails: These are applied to the inputs and outputs of custom actions (also known as tools) that the LLM needs to invoke.
- Output rails: These are applied to the LLM's generated outputs. An output rail can either refuse the output, blocking it from being sent to the user or modify it (such as by erasing sensitive data).

GitHub - NVIDIA/NeMo-Guardrails: NeMo Guardrails is an open-source toolkit for easily adding programmable guardrails to LLM-based conversational systems.

nemo-guardrails-llamaindex-rag/nemo_guardrails_llamaindex_rag.ipynb at main · wenqiglantz/nemo-guardrails-llamaindex-rag

Retrievers Comparsion

Retriever	Advantages	Limitations
BM25 What Is BM25 (Best Match 25): Full Breakdown - Luigi's Box .	Dynamic ranking Effective for long queries Scalability Simplicity	No semantic understanding No personalization Sensitivity to term frequency
Amazon Kendra Pros and Cons of Amazon Kendra 2024	Improved search efficiency AI-based technology Structured data retrieval Enhanced user experience Optimized for business use	Integration challenges High cost
ActiveLoop Deep Memory Activeloop Deep Memory | 🦜🔗 LangChain	Enhanced accuracy Reduced cost	
Arcee Retriever Arcee.ai on LinkedIn: GitHub - arcee-ai/DALM: Domain Adapted Language Modeling Toolkit - E2E RAG	Enhanced performance Simplicity and effectiveness	Dependency on generator performance Complexity and resource intensiveness Overfitting
ArxivRetriever	Helps to retriever scientific articles	
Amazon Bedrock	Streamlined workflow Data customization Automated ingestion workflow Scalable infrastructure	
BREEBS	Reduces hallucinations Give access to sources Provides seamless context	
Chaindesk	Access to data from anywhere. Can be connected with chatgpt through plugin or other llm.	
ChatGPT Plugin	Retrieve real time information	

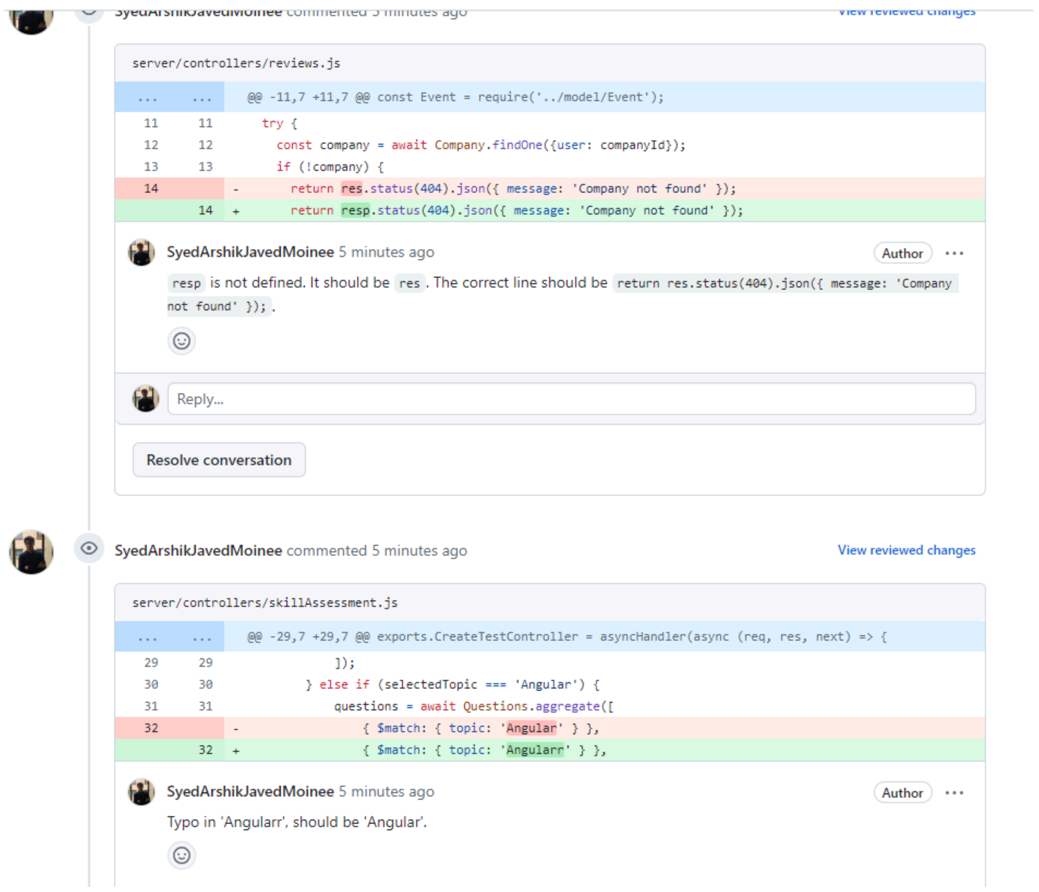
	Retrieve knowledge base information Perform actions on behalf of the user	
DocArray	Manages multi-modal data Flexible to store and search	
Elasticsearch BM25	It provides a distributed, RESTful search engine with an HTTP web interface and JSON documents. Supports keyword search, vector search, hybrid search, and complex filtering.	
Elasticsearch	Enable flexible access to Elasticsearch features through Query DSL.	
Embedchain	Streamlines the creation of personalized LLM applications, managing unstructured data efficiently. It efficiently segments data, generates relevant embeddings, and stores them in a vector database. Offers diverse APIs for contextual information extraction, precise answers, and interactive chats.	
Fleet AI Context	Provides high-quality embeddings of the top 1200 Python Libraries & their documentation. Enables embedding the world's important data, starting with the	

	top 1200 Python libraries.	
Google Vertex AI	Google Vertex AI Search allows quick building of generative AI-powered search engines using semantic search.	

Context Window Solution

One solution for this problem can be that we can set a variable like window_size = 10, then we will retrieve 10 lines of code before the changed code and 10 lines of code after the changed code. Obviously, the higher this value will be, more context will be retrieved which will eventually help the LLM in making a good review.

Given below are the reviews in which only context window has been passed, not even the context of the repo has been passed in this.



SyedArshikJavedMoinee commented 5 minutes ago View reviewed changes

server/controllers/reviews.js

```
@@ -11,7 +11,7 @@ const Event = require('../model/Event');
   try {
     const company = await Company.findOne({user: companyId});
     if (!company) {
-      return res.status(404).json({ message: 'Company not found' });
+      return resp.status(404).json({ message: 'Company not found' });
```

SyedArshikJavedMoinee 5 minutes ago Author

`resp` is not defined. It should be `res`. The correct line should be `return res.status(404).json({ message: 'Company not found' });`.

Reply...

Resolve conversation

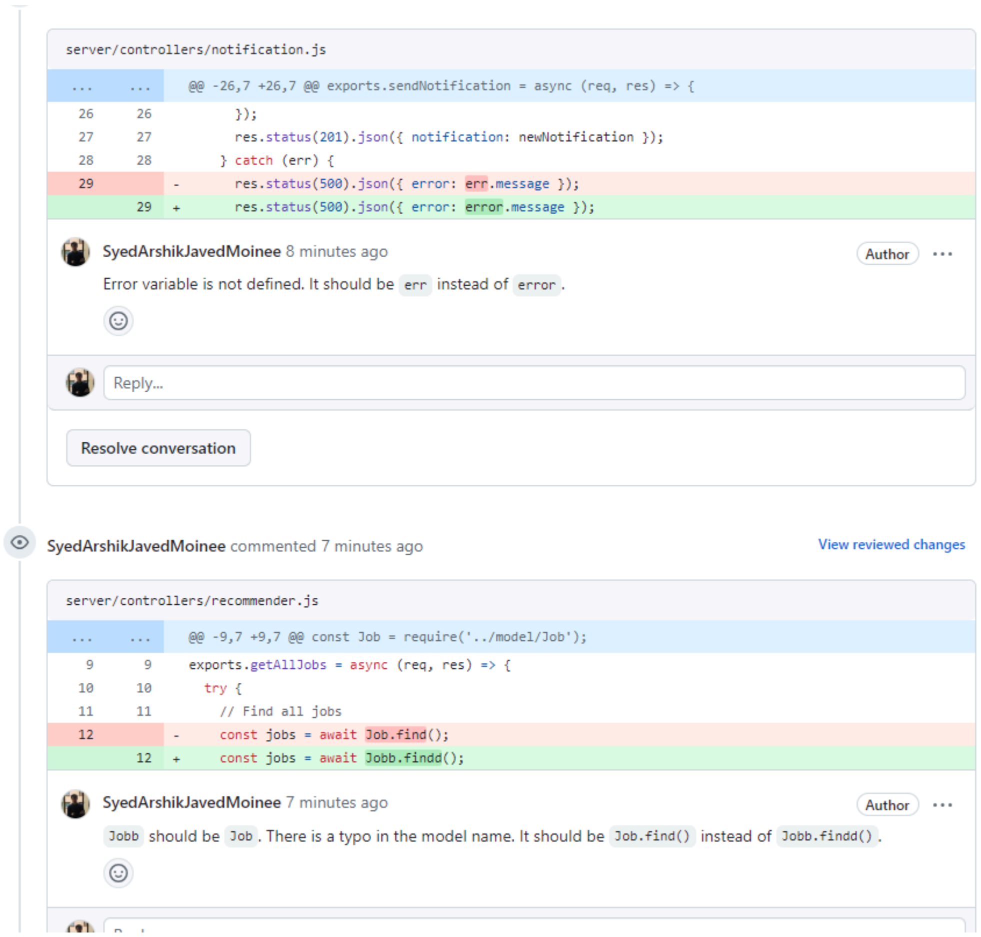
SyedArshikJavedMoinee commented 5 minutes ago View reviewed changes

server/controllers/skillAssessment.js

```
@@ -29,7 +29,7 @@ exports.CreateTestController = asyncHandler(async (req, res, next) => {
           ]);
       } else if (selectedTopic === 'Angular') {
           questions = await Questions.aggregate([
-              { $match: { topic: 'Angular' } },
+              { $match: { topic: 'Angularr' } },
```

SyedArshikJavedMoinee 5 minutes ago Author

Typo in 'Angularr', should be 'Angular'.



server/controllers/notification.js

```
@@ -26,7 +26,7 @@ exports.sendNotification = async (req, res) => {
          });
          res.status(201).json({ notification: newNotification });
        } catch (err) {
-         res.status(500).json({ error: err.message });
+         res.status(500).json({ error: error.message });
```

SyedArshikJavedMoinee 8 minutes ago Author

Error variable is not defined. It should be `err` instead of `error`.

SyedArshikJavedMoinee commented 7 minutes ago

server/controllers/recommender.js

```
@@ -9,7 +9,7 @@ const Job = require('../model/Job');
  exports.getAllJobs = async (req, res) => {
    try {
      // Find all jobs
-     const jobs = await Job.find();
+     const jobs = await Jobb.findd();
```

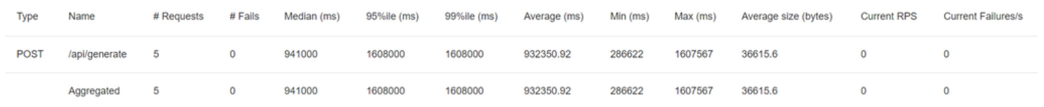
SyedArshikJavedMoinee 7 minutes ago Author

`Jobb` should be `Job`. There is a typo in the model name. It should be `Job.find()` instead of `Jobb.findd()`.

Stress Testing EC2 Endpoint

We have deployed Llama 3-70b using Ollama on a g4dn.4xlarge EC2 instance. To test its response time and concurrency, we are using Locust, a load testing tool. We are making API calls to the model's endpoint and simulating 20 concurrent users with a spawn rate of 5 over a 30-minute period.

The results of this evaluation are as follows:



Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
POST	/api/generate	5	0	941000	1608000	1608000	932350.92	286622	1607567	36615.6	0	0
	Aggregated	5	0	941000	1608000	1608000	932350.92	286622	1607567	36615.6	0	0

Results for Experiment 1:

- **Number of Requests:** 5 requests were made in total.
- **Failures:** There were 0 failures observed.
- **Average Time taken to complete a request is:** 15.53 minutes
- **Minimum Time taken to complete a request is:** 4.77 minutes
- **Maximum Time taken to complete a request is:** 26.79 minutes

Overall, while the endpoint functioned without failures during this limited test, its performance under higher loads might lead to longer response times, potentially impacting user experience.

Results for Experiment 2:

500 users with a spawn rate of 0.139

Number of Requests: 11 requests were made in total.

Failures: There were 0 failures observed.

Average time taken to complete a request: 33.09 minutes

Minimum time taken to complete a request: 6.49 minutes

Maximum time taken to complete a request: 57.29 minutes

Again the endpoint functioned without failures during this test, its performance under higher loads might lead to longer response times, potentially impacting user experience.

Results for Experiment 3:

500 users with a spawn rate of 0.139

Number of Requests: 405 requests were made in total.

Failures: There were 11 failures observed.

Average time taken to complete a request: 15.89 minutes

Minimum time taken to complete a request: 11.87 seconds

Maximum time taken to complete a request: 31.72 minutes

Now in this, the endpoint functioned with a relatively low failure rate but demonstrated significant variability in response times.

Literature Review of RAG/LLM Based Log Analysis or Incident Management or Root Cause Analysis

Paper Title	**Root Cause Analysis**	**Link**
Empirical Evaluation of LLM-Based Agent for Root Cause Analysis in Cloud Incident Management	Cloud Incident Management	Literature Review of RAG/LLM Based Log Analysis or Incident Management or Root Cause Analysis
Automatic Root Cause Analysis via Large Language Models for Cloud Incidents	Cloud Incident	Automatic Root Cause Analysis via Large Language Models for Cloud Incidents
RAGLog: Log Anomaly Detection using Retrieval Augmented Generation	Log Anomaly Detection	RAGLog: Log Anomaly Detection using Retrieval Augmented Generation
Xpert: Empowering Incident Management with Query Recommendations via Large Language Models	KQL Query Generation for Log Analysis	https://arxiv.org/pdf/2312.11988
Knowledge-aware Alert Aggregation in Large-scale Cloud Systems: a Hybrid Approach	Alert aggregation on Cloud System Failure	https://arxiv.org/pdf/2403.06485
PACE-LM: Prompting and Augmentation for Calibrated Confidence Estimation with GPT-4 in Cloud Incident Root Cause Analysis	Confidence Estimation of Generated Cause	https://arxiv.org/pdf/2309.05833
Dependency Aware Incident Linking in Large Cloud Systems	Identify Dependencies of Faults	https://arxiv.org/pdf/2403.18639

Summary of the Literature Review:

Introduction

With the growing complexity of AI systems, prompt resolution of incidents is crucial to maintaining system capabilities. Traditional methods like on-call engineers require extensive expertise, skill set and time [1][2] . It is also prone to errors and is very complex and challenging even for skilled and expert personas [2]. Root cause analysis (RCA) in cloud incident management causes significant challenges due to the need for specialised data and information gathering [1]. Different approaches are used for RCA including [1] focuses on evaluating the effectiveness of LLM-based agents, specifically ReAct, in RCA tasks without fine-tuning, and explores practical considerations for their real-world application. In [2], RCA-Copilot is a tool empowered by LLM for automating RCA of cloud services incidents. Another approach proposed in [3], helps in the detection of log anomalies from system logs as it is one of the important activities to ensure cyber resilience of a system. It also helps in fault detection. In [3], a RAG model is built to detect log anomalies from system log data.

Technique and Tools

Different tools and techniques are used in different researches to solve this problem. In [1], ReAct, an LLM-based agent utilising OpenAI's GPT-4 8K as the base LLM and GPT 3.5 Turbo for summarization, is employed for RCA tasks. The agent utilises various tools within the Langchain framework, including ToolFormer for tool usage modelling, ReACT Loop for planning, and Incident detail for question answering. Historical Incidents serve as a retrieval tool, along with a Sentence Transformer retriever and Database query tool for data retrieval and analysis. Human interaction tools are also integrated into the workflow. An RCA-Copilot was proposed in [2]. RCA-Copilot operates in two stages: Diagnostic Information Collection and Root Cause Prediction. The techniques include incident handlers tailored to specific alert types, multi-source data collection, and LLM for incident explanation. Actions in incident handlers include scope switching, query action, and mitigation action, controlled by Decision Tree's flow. Also used Nearest Neighbor Search and FastText embedding model aid in contextual semantics extraction and prompt construction for LLM. In [3], a RAG model is utilized for analysing log entries. Vector database is used a store and LLM is used for semantic analysis.

Challenges

There are several challenges associated with automating RCA. Obtaining relevant information for RCA, especially from specialised sources like logs and monitoring services, presents a significant challenge [1]. Crafting queries for these domains and processing tabular data require specialised skills [1]. Additionally, determining which supplementary data to collect and how to analyse it, including information not present in incident reports, adds complexity to the RCA pipeline [1][2]. Also, overwhelming volume of Troubleshoot Guides (TSGs) makes identifying relevant solutions time-consuming [2]. Sometimes, these models also require high resource consumption and also faces execution latency when running LLM and performing log analysis [3].

Evaluation Metrics

Different evaluation metrics were used in different techniques. In [1], three lexical similarity metrics (BLEU, METEOR, Rouge) and one semantic similarity metric (Berts) are utilised for evaluation. In [2], Micro-f1 and Macro-f1 were used for predicting root causes. In [3], the researchers used precision, recall and F1 score for evaluation.

Public Dataset:

LEMMA RCA: 🤗 Lemma-RCA-NEC (Lemma RCA)

For the IT domain's microservice platforms (Product Review Platform):

- **Platform Composition**: Six OpenShift nodes (such as ocp4-control-plane-1 through ocp4-control-plane-3, ocp4-compute-1 and ocp4-compute-2, and ocp4-infra-1) and 216 system pods (including ProductPage, MongoDB, review, rating, payment, Catalogue, shipping, etc.).
- **Fault Simulations**: Four distinct system faults simulated on four different dates, including out-of-memory, high-CPU-usage, external-storage-full, and DDoS attack.
- **Metrics**: Recorded 11 types of node-level metrics (e.g., net disk IO usage, net disk space usage, etc.) and six types of pod-level metrics (e.g., CPU usage, memory usage, etc.) with a time granularity of 1 second using Prometheus.
- **Logging**: Collected log data using ElasticSearch with detailed timestamps and retrieval periods, including timestamp, pod name, and log message.
- **System Status Information**: Collected using JMeter, including elapsed time, latency, connect time, thread name, throughput, etc. Latency is considered a Key Performance Indicator (KPI).

For the Cloud Computing Platform:

- **Fault Simulations**: Six different types of faults simulated on eleven system nodes (such as cryptojacking, mistakes made by GitOps, configuration change failure, etc.)
- **Metrics**: System metrics extracted directly from CloudWatch Metrics on EC2 instances, with a time granularity of 1 second.
- **Logs**: Acquired from CloudWatch Logs, consisting of log messages, api debug log, and mysql log.
- **KPIs**: Tracked latency, error rate, and utilisation rate using JMeter.

Literature Review of Each Technique

Empirical Evaluation of LLM-Based Agent for Root Cause Analysis in Cloud Incident Management

Link: Exploring LLM-based Agents for Root Cause Analysis

Introduction

With the growing complexity of AI systems, prompt resolution of incidents is crucial to maintaining system capabilities. Traditional methods like on-call engineers require extensive expertise, skill set and time. Root cause analysis (RCA) in cloud incident management causes significant challenges due to the need for specialised data and information gathering. This paper focuses on evaluating the effectiveness of LLM-based agents, specifically ReAct, in RCA tasks without fine-tuning, and explores practical considerations for their real-world application.

Technique and Tools

ReAct, an LLM-based agent utilising OpenAI's GPT-4 8K as the base LLM and GPT 3.5 Turbo for summarization, is employed for RCA tasks. The agent utilises various tools within the Langchain framework, including ToolFormer for tool usage modelling, ReACT Loop for planning, and Incident detail for question answering. Historical Incidents serve as a retrieval tool, along with a Sentence Transformer retriever and Database query tool for data retrieval and analysis. Human interaction tools are also integrated into the workflow.

Challenges

Obtaining relevant information for RCA, especially from specialised sources like logs and monitoring services, presents a significant challenge. Crafting queries for these domains and processing tabular data require specialised skills. Additionally, determining which supplementary data to collect and how to analyse it, including information not present in incident reports, adds complexity to the RCA pipeline.

Goal

The study aims to assess the effectiveness of LLM agents for RCA tasks without fine-tuning and identify practical considerations for their implementation in real-world scenarios.

Summary of Techniques

Zero-shot prompting is employed with ReAct, leveraging techniques like Retrieval Baseline (RB), Retrieval Augmented Generation (RAG), Chain of Thought (CoT), and Interleaving Retrieval - Chain of Thought (IR-CoT) for enhanced reasoning abilities and information retrieval.

Evaluation Metrics

Three lexical similarity metrics (BLEU, METEOR, Rouge) and one semantic similarity metric (Berts) are utilised for evaluation.

Conclusion

The empirical evaluation of ReAct demonstrates competitive performance in out-of-domain, zero-shot scenarios compared to strong baselines like RAG and CoT, with lower rates of factual inaccuracies. Discussion comments from incident reports have very minimum impact on ReAct's performance which indicates limitations in RCA on static datasets. A case study showcases the potential of LLM-based agents for autonomous RCA in real-world settings. Future directions include building simulated RCA environments to overcome dataset limitations and advance incident management practices and decision-making automation in software engineering.

Title: *Automatic Root Cause Analysis via Large Language Models for Cloud Incidents*

Link: https://arxiv.org/pdf/2305.15778

1. **Introduction**
   Traditional methods used for RCA relies on manual investigations therefore it's prone to errors and is very complex and challenging even for the trained and skill personas. RCA-Copilot is a tool empowered by LLM for automating RCA of cloud services incidents. The accuracy is up to 0.766 over the dataset of transport service in Microsoft.

1. **Challenges**
   - Lack of necessary information for accurate root cause analysis.
   - Noisy, incomplete, and inconsistent data.

- Overwhelming volume of Troubleshoot Guides (TSGs) makes identifying relevant solutions time-consuming.
- Ever-evolving cloud systems may render TSGs insufficient for new problems.

2. **Objective**
   - Develop an automated tool for cloud incident root cause analysis.
   - Assist Operational Center Engineers (OCEs) in constructing incident-specific automatic workflows for efficient data collection.
   - Integrate LLM within RCA-Copilot to autonomously predict incident root cause categories and generate explanations.
3. **Tools & Techniques**
   - RCA-Copilot operates in two stages: Diagnostic Information Collection and Root Cause Prediction.
   - Techniques include incident handlers tailored to specific alert types, multi-source data collection, and LLM for incident explanation.
   - Actions in incident handlers include scope switching, query action, and mitigation action, controlled by Decision Tree's flow.
   - Nearest Neighbor Search and FastText embedding model aid in contextual semantics extraction and prompt construction for LLM.

For diagnostic information collection, an incident handler is created for each alert type to collect diagnostic information. RCA-Copilot incident handler is a workflow that comprises of series of actions. OCE's can build and modify these handlers. The main three actions included in these handlers are
1-scope switching: Helps in adjusting the scope of the data collection based on the specific needs of the incident.
2- query action: Can gather data from various sources.
3- mitigation action: Helps in suggesting strategic steps to control an incident.
The incident handler uses Decision Tree's control flow for decision making.
Handler action: RCA-Copilot uses multi-source data. Predefined actions are used to collect relevant diagnostic information through incident handler. LLM for incident explanation: Each incident is manually assigned a root cause category by an expert OCE.
Chain of thought prompting allows LLMs to generate intermediate reasoning steps that lead to the final answer. Embedding Model: To extract the contextual semantics, FastText embedding model is used.
Nearest Neighbor Search: To combine past incidents as samples in the prompt, nearest neighbor search is used with a new similarity formula that computes Euclidean distance for every pair of incident vector and then considers the temporal distance between incidents.
Diagnostic information summary: Another layer to use LLM's ability to summarize the diagnostic information before making the reasoning.
Prompt construction: LLM summarizes the diagnostic information which forms the basis for the COT prompting.

1. **Evaluation**
   - Metrics used for evaluation: Micro-F1 and Macro-F1 for predicting root causes.
   - Compared approaches: XGBOOST, FastText, Fine-Tune GPT, GPT-4 Prompt, GPT-4 Embed.
2. **Conclusion**
   RCA-Copilot is a tool for cloud incident management that helps in efficient root cause analysis for OCEs. It uses data from multi-sources through diagnostic collection stage by using predefined incident handlers. After this, RCA-Copilot integrates LLM to predict the root cause.
   The model automatically process the collected diagnostic data and also does the prediction of the root cause.

Title: *RAGLog: Log Anomaly Detection using Retrieval Augmented Generation*

Link: https://arxiv.org/pdf/2311.05261

- **Introduction:**
  - Detection of log anomalies from system logs is one of the important activities to ensure cyber resilience of a system. It helps in fault detection. A RAG model is built to detect log anomalies from system log data.
- **Challenge with Log Analysis:**
  - As logs belong to different systems, identifying faults from a large volume of logs becomes very challenging.
- **Proposed Model and Tools:**
  - A RAG model is utilised for analysing log entries.
  - Vector db is used as a store.
  - LLM is used for performing semantic analysis between retrieved log samples.
- **Datasets and Evaluation Metrics:**

    - Datasets used: BGL and Thunderbird.
    - Evaluation Metrics: Precision, Recall, F1 Score are used to evaluate the performance of the log anomaly detection system.
- **Experimental Approaches:**
    - Two approaches were used for experimentation:
        - First Approach: Populating the database with randomly selected samples from log datasets containing normal log entries.
        - Second Approach: Populating the database with selected samples from the log database using K-Means clustering algorithm on normal log entries.
- **Experimental Results:**
    - Precision score: 0.91
    - Recall score: 0.88
    - F1 score: 0.89
- **Challenges Identified:**
    - High resource consumption and execution latency observed when running LLM and performing log analysis.

Title: Xpert: Empowering Incident Management with Query Recommendations via Large Language Models

Link: https://arxiv.org/pdf/2312.11988

Main Idea

To automate the KQL query recommendation process by employing a domain-specific language for incident management in a large-scale cloud management system at Microsoft.

Detailed Methodology

1. **Information Collection:**
    - Data collected includes metadata, incident title, summary, and discussions.
2. **Data Pre-processing:**
    - Repetitive information is removed.
    - If incident context exceeds a token threshold, the sample is clipped.
3. **KQL Query Recommendation:**
    - Combines in-context learning and large language model (LLM) capabilities.
    - Three-step generation process:
        - Similar incident retrieval
        - Prompt construction
        - DSL query generation
4. **Post Processing:**
    - Includes:
        - **Post Validator:** Checks grammar and syntax using an intrinsic compiler abstract syntax tree.
        - **Post Rectifier:** Corrects invalid queries.
5. **Dataset:**
    - KQL queries from the top 10 services with the highest incident volume at Microsoft.
6. **Evaluation:**
    - XPERT compared with BART, T5, CODET5, CODET5+ using the following evaluation metrics:
        - SacreBLEU
        - METEOR
        - Xcore
        - TableACC
        - Identicality

        - Validity

How XPERT Works

- Upon receiving a new incident ticket:
    - Data processor gathers incident information.
    - Incident context is preprocessed into a format suitable for an LLM.
- An embedding model vectorizes the incident context.
- Historical incidents and their KQL queries are searched based on similarity.
- Retrieved information is combined with the target incident context.
- Input is fed to the LLM to generate an initial KQL template and query.
- Post validation process checks query validity:
    - Valid queries are recommended to OCEs for review.
    - Invalid queries undergo rectification.
- Recommended queries crafted by OCEs are added to the vector database for future use.

Title: Knowledge-aware Alert Aggregation in Large-scale Cloud Systems: a Hybrid Approach

Link: https://arxiv.org/pdf/2403.06485

Main Idea

A hybrid approach called COLA is introduced to address limitations in tracing system failures or alerts using semantic similarity or statistical methods. COLA leverages external knowledge (Standard Operation Procedures - SOPs) and combines correlation mining with Large Language Models (LLM) for online alert aggregation.

Methodology

1. **Actual Failure:**
    - When a failure occurs and a threshold of failed requests is reached, an alert is triggered to OCEs.
2. **Input Data:**
    - Alerts and corresponding Standard Operation Procedures (SOPs).
3. **Output Data:**
    - Grouped alerts where alerts with the same cause are correlated, reducing the number of alerts and facilitating efficient understanding.
4. **First Step: Pre-processing**
    - Alerts are pre-processed based on physical regions ("isolated systems").
    - Alerts are divided based on arrival time.
5. **Second Step: Correlation Mining**
    - Utilizes temporal and spatial relationships to determine correlations between alerts.
    - Outputs a similarity score from correlation analysis.
6. **Third Step: LLM Reasoning**
    - Uses two LLM techniques: In Context Learning and Supervised Fine Tuning.
    - Learns domain knowledge and performs reasoning tasks.

Evaluation

- **Evaluation Metrics:** F1 score.
- **Comparison:** COLA was compared with:
    - Machine learning algorithms: FP-Growth, DBSCAN.

- Research frameworks: AlertStorm, LiDAR, OAS, iPACK.

Conclusion

The proposed COLA framework integrates correlation mining and LLM reasoning components effectively, providing a robust solution for handling large volumes of alerts in practical scenarios. This approach enhances alert aggregation efficiency, reduces time costs, and improves understanding of system failures or alerts.

Title: PACE-LM: Prompting and Augmentation for Calibrated Confidence Estimation with GPT-4 in Cloud Incident Root Cause Analysis

Link: https://arxiv.org/pdf/2309.05833

Main Idea

Many cloud providers utilize AI models to predict the root cause of failures, but these predictions often exhibit discrepancies and can be unreliable ("hallucinations"). To address this, researchers proposed a model that estimates the confidence score of predicted root causes to ensure they are grounded in reality rather than being hallucinatory.

Methodology

1. **Method Overview:**
   - The method involves two scoring phases:
     - **LLM-based Confidence Estimator:** Evaluates confidence in making judgments.
     - **Historical References Rating:** Rates root cause predictions based on historical data.
   - An optimization step combines these scores to assign a final confidence level.
2. **First Step: Retrieval Augmented Procedure (RAG)**
   - **Data Retrieval:** Retrieves a list of similar historical incidents relevant to each query using semantic similarity-based dense retrievers.
   - Helps identify past incidents with similar root causes.
3. **LLM-based Confidence Evaluation:**
   - **Input Preparation:** Historical incidents and associated root causes are fed into an LLM.
   - **Analysis:** The LLM assesses whether it has sufficient information to determine the underlying root cause of the current incident.
   - **Root Cause Prediction:** Based on historical data, the LLM generates a root cause prediction for the current incident.
4. **Final Confidence Assignment:**
   - **Combination of Scores:** COE (Confidence of Evaluation) and RCE (Root Cause Evaluation) scores are combined to derive the final confidence score.
   - **Optimization:** An optimization step refines the final confidence assignment.

Dataset

- **Data Source:** Incident data from various services with varying levels of severity.
- **Labels Generation:** Utilized GPT-4.
- **Evaluation Metrics:**
  - F1 score
  - Reliability diagrams
  - ECE (Expected Calibration Error) score
  - Human evaluation

Conclusion

The proposed approach divides the confidence estimation process into two phases:

- **Strength of Evidence:** Measures the evidence strength from historical references.

- **Model Prediction Quality:** Quantifies the quality of the model's prediction. Through an optimization step, final confidence assignments are derived from these two scores. This method enhances the reliability of root cause predictions in cloud environments by grounding them in historical context and model evaluation.

Title: Dependency Aware Incident Linking in Large Cloud Systems

Link: https://arxiv.org/pdf/2403.18639

Main Idea

System failures are common in technical services, and resolving them quickly is crucial. However, a single failure can sometimes trigger multiple downstream failures across different dependent services. To address this challenge, researchers developed a dependency-aware incident linking framework. This framework utilizes both textual information and service dependency graph data to enhance the accuracy and coverage of incident links, not only within the same service but also across different services.

Methodology

1. **Input Data:**
    - **Textual Description:** Includes title, topology, and other descriptive elements.
    - **Categorical Information:** Includes monitor ID, failure type, owning team ID, and other categorical attributes.
    - **Service Structural Information:** Includes dependency graph representation of services.
2. **Output Data:**
    - Generates a final label indicating the relationship between incidents, such as Duplicate, Related, Responsible, etc., based on similarity scores.
3. **Framework Details:**
    - **Dependency-Aware Incident Linking:**
        - Utilizes textual descriptions and categorical information to understand incident contexts.
        - Incorporates service dependency graph information to identify relationships between incidents across different services.
4. **Learning Relationship Between Incidents:**
    - **Learning Process:**
        - Learns patterns and relationships between incidents based on input data.
        - Uses supervised learning or other techniques to infer incident links.
5. **Output Label Generation:**
    - **Final Label Assignment:**
        - Computes similarity scores between incidents.
        - Assigns final labels (Duplicate, Related, Responsible, etc.) based on the similarity score thresholds.

Conclusion

The dependency-aware incident linking framework represents a significant advancement in incident management. By integrating textual descriptions, categorical information, and service dependency graphs, it improves the accuracy and coverage of incident links. This approach not only aids in resolving incidents within the same service but also facilitates quicker identification and resolution of issues that span across multiple dependent services.

RAG Failure Causes

Title: Uptrain RCA for RAG

Link: uptrain/examples/root_cause_analysis/rag_with_citation.ipynb at main · uptrain-ai/uptrain

Following are the 5 failure points that are identified in uptrain RCA

1. Poor Context Utilisation: The LLM is not able to utilize all the information present in the context
2. Poor Retrieval: The context given to an LLM does not have information relevant to the question
3. Hallucination: The generated response is not supported by information present in the context
4. Incorrect Citation: While the response is correct, it's not correctly cited, eroding users' trust
5. Incomplete Question: The user's question itself is unclear or does not make sense.

Title: Galileo io

Link: Mastering RAG: How To Observe Your RAG Post-Deployment - Galileo

- **Chain Execution Information**:
  - Understanding the execution of processing chains, particularly in Langchain LLM systems, is crucial.
  - It involves tracking data and operations flow within the chain, from context retrieval to response generation.
  - This helps in comprehending system behavior and pinpointing potential points of failure.
- **Retrieved Context**:
  - Observing context retrieved from an optimized vector database is essential.
  - It helps assess the relevance and adequacy of information provided to the language model.
  - This includes tracking the retrieval process, including context selection and presentation to the model.
- **ML Metrics**:
  - ML metrics offer insights into the performance and behavior of the language model.
  - They encompass aspects such as adherence to context, indicating how well the model maintains relevance and coherence in its responses.
- **System Metrics**:
  - System metrics provide insights into the operational health and performance of the RAG (Retrieval-Augmented Generation) deployment infrastructure.
  - They cover aspects like resource utilization, latency (response time), and error rates.
  - These metrics are crucial for assessing the overall efficiency and reliability of the deployed system.

Monitoring Retrieval-Augmented Generation (RAG) systems involves monitoring various metrics to detect potential issues. Setting alerts based on these metrics enables AI teams to actively oversee system performance and promptly tackle any emerging issues.

Title: Seven Failure Points When Engineering a Retrieval Augmented Generation System

Link: https://arxiv.org/pdf/2401.05856

Following are the seven failure points of a RAG system

1. Missing Content: Failure can occur while posing a question that cannot be addressed using the existing documents.
2. Missed Top Ranked: The document contains the answer to the question but didn’t rank high enough to be presented to the user.
3. Not in Context: Documents containing the answer were successfully retrieved from the database but were not included in the context used to generate a response.
4. Wrong Format: The question required extracting information in a specific format, such as a table or list, yet the large language model disregarded this instruction.

5. Incorrect Specificity: The response includes an answer, but it lacks the required specificity or is overly specific, failing to meet the user's needs.
6. Not Extracted: The answer is within the context provided, but the large language model fails to accurately extract it.
7. Incomplete: Incomplete answers are not necessarily incorrect but lack some information, even though it was present in the context and could have been extracted.

Failure	**Lesson**	**Description**	**Reference to Case Study**
FP	**Lesson**	**Description**	**Case Studie**s
FP4	Larger context get better results (Context refers to a particular setting or situation in which the content occurs)	A larger context enabled more accurate responses (8K vs 4K). Contrary to prior work with GPT-3.5 (Liu et al., 2023b)	AI Tutor
FP1	Semantic caching drives cost and latency down	RAG systems struggle with concurrent users due to rate limits and the cost of LLMs. Prepopulate the semantic cache with frequently asked questions (Bang, 2023).	AI Tutor
FP5-7	Jailbreaks bypass the RAG system and hit the safety training.	Research suggests fine-tuning LLMs reverses safety training (Lermen et al., 2023), test all fine-tuned LLMs for RAG system.	AI Tutor
FP2, FP4	Adding meta-data improves retrieval.	Adding the file name and chunk number into the retrieved context helped the reader extract the required information. Useful for chat dialogue.	AI Tutor
FP2, FP4-7	Open source embedding models perform better for small text.	Opensource sentence embedding models performed as well as closed source alternatives on small text.	BioASQ, AI Tutor
FP2-7	RAG systems require continuous calibration.	RAG systems receive unknown input at runtime requiring constant monitoring.	AI Tutor, BioASQ
FP1, FP2	Implement a RAG pipeline for configuration.	A RAG system requires calibrating chunk size,	Cognitive Reviewer, AI Tutor, BioASQ

		embedding strategy, chunking strategy, retrieval strategy, consolidation strategy, context size, and prompts.	
FP2, FP4	RAG pipelines created by assembling bespoke solutions are suboptima.	End-to-end training enhances domain adaptation in RAG systems (Siriwardhana et al., 2023).	BioASQ, AI Tutor
FP2-7	Testing performance characteristics are only possible at runtime.	Offline evaluation techniques such as G-Evals (Liu et al., 2023a) look promising but are premised on having access to labelled question and answer pairs.	Cognitive Reviewer, AI Tutor

GraphRAG

Paper Link: https://arxiv.org/pdf/2404.16130

Github Repository: GitHub - microsoft/graphrag: A modular graph-based Retrieval-Augmented Generation (RAG) system

GraphRAG Flowchart: https://tinyurl.com/2s4ytcur

Limitations of Basic RAG:

- Baseline RAG struggles to connect the dots.
- Baseline RAG performs poorly when being asked to holistically understand summarized semantic concepts over large data collections or even singular large documents.

GraphRAG Pipeline:

- **Source documents:** Deciding how finely to split input texts from source documents into chunks for processing, balancing efficiency with the quality of information extraction.
- **Text chunks:** Extracting smaller or larger portions of text impacts the efficiency and accuracy of extracting entity references, balancing between recall and precision in the extraction process.
- **Element instances:** Identifying and extracting instances of graph nodes and edges from text chunks using tailored prompts, including domain-specific examples for enhanced accuracy.
- **Element Summaries:** Using an LLM to generate descriptive summaries of entities, relationships, and claims extracted from texts, ensuring coherence despite variations in entity references.
- **Graph communities:** Creating an undirected graph where nodes represent entities and edges their relationships, employing community detection algorithms like Leiden to reveal hierarchical structures.
- **Community summaries:** Generating summaries of node communities in hierarchical levels, aiding in understanding dataset themes and structures, essential for query-focused summarization.
- **Community answers:** Generating intermediate answers from community summaries for user queries, evaluating and integrating them based on relevance and helpfulness scores.
- **Global answer:** Compiling a final answer to a user query from the integrated community answers, ensuring comprehensive coverage and relevance.

Datasets:

Podcasts: Compiled transcripts of podcast conversations between Kevin Scott, Microsoft CTO, and other technology leaders (Behind the Tech, Scott, 2024). Size: 1669 × 600-token text chunks, with 100-token overlaps between chunks (∼1 million tokens)

News Articles: Benchmark dataset comprising news articles published from September 2013 to December 2023 in a range of categories, including entertainment, business, sports, technology, health, and science (MultiHop-RAG; Tang and Yang, 2024). Size: 3197 × 600-token text chunks, with 100-token overlaps between chunks (∼1.7 million tokens).

Community Levels:

Four levels of communities are included.

- CO: Uses root-level community summaries (fewest in number) to answer user queries.
- C1: Uses high-level community summaries to answer queries. These are sub-communities of C0, if present, otherwise C0 communities projected down.
- C2: Uses intermediate-level community summaries to answer queries. These are subcommunities of C1, if present, otherwise C1 communities projected down.
- C3: Uses low-level community summaries (greatest in number) to answer queries. These are sub-communities of C2, if present, otherwise C2 communities projected down.

Evaluation Metrics:

- Comprehensiveness.
- Diversity.
- Empowerment.
- Directness.

Use cases of GraphRAG

1- Critical Information Discovery

2- Analysis

3- Incident Management

4- Health care

Link:https://www.linkedin.com/pulse/enhancing-intelligent-applications-using-graphrag-suresh-mandava-pgk3c/

Multimodal RAG & Open Source Embedding Models

Comparison of Embedding Models for Different Data Types & Strategies for Selecting Chunking Techniques

Considerations Before Chunking:

- Structure & Length of Documents:
- Long documents: Books, academic papers, etc.
-
- Short documents: Social media posts, customer reviews, logs, etc.
- Embedding Model:
- Chunk size determines which embedding model should be used.
- Expected Queries:
- What is the use case?

Chunking strategies are composed of three key components — splitting technique, chunk size, and chunk overlap. Picking the right strategy involves experimenting with different combinations of the three components.

Unstructured | The Unstructured Data ETL for Your LLM can be used for multimodal chunking.

Link: Chunking PDFs and Multimodal Documents: Efficient Methods for Handling Text, Tables, and Images for…

NLTK Sentence Tokenizer:

Useful for splitting text into sentences.

Key points:

Language dependent (English)

Can misinterpret abbreviations or punctuations.

It doesn’t consider the semantic relationship between sentences.

Spacy Sentence Splitter:

Useful for splitting text into smaller sentences.

Key points:

Can misinterpret abbreviations or punctuations.

Strictly adheres to sentence boundaries and is useful in smaller text units.

Langchain Character Text Splitter:

It recursively divides the text at specific characters.

It can be customized by changing the chunk size and overlap parameters according to the needs.

Considerations before selecting an embedding models:

Considerations for selecting a model include its size and memory usage, which affect computational resources and latency. Larger models may improve retrieval performance but can increase latency and risk overfitting. Starting with a smaller, efficient model allows for faster iteration and building a baseline.

Embedding dimensions refer to the length of vector representations; larger dimensions capture more detail but may not always be necessary. For tasks like RAG applications, embedding typically covers about a paragraph of text (e.g., max tokens of 512). Special cases may require models with larger context windows for longer texts.

Embedding Models:

- PowerPoint Data (multimodal):

Needs multimodal embedding models like CLIP, SIGCLIP or ImageBind.

CLIP embeddings to improve multimodal RAG with GPT-4 Vision | OpenAI Cookbook

Get multimodal embeddings | Generative AI on Vertex AI | Google Cloud

Multimodal Retrieval Augmented Generation(RAG) | Weaviate

Guide to Multimodal RAG for Images and Text | by Ryan Siegler | KX Systems | Medium

- PDF or Text File Data:
- Need text embedding models like Voyage
- Jina AI: Supports PDFs with text and images (can be tested with ppt)
- Voyage AI: Supports PDFs with text.
- Open AI Embedding Model: Supports PDFs with text.
- Visualized BGE: Support Multimodal Data
- Cohere Embedding: Supports Emails, Invoices, CVs, Support Tickets, Log Messages, and Tabular Data
- Olama Embedding: Structured Data Extraction from Images, Retrieval Augmented Image Captioning and Multi modal RAG.

RAG Pipeline for Multimodal Data:

Beyond Text: Taking Advantage of Rich Information Sources With Multimodal RAG).

An Easy Introduction to Multimodal Retrieval-Augmented Generation | NVIDIA Technical Blog

Challenges in Working With Multimodal Data:

Representing information across different modalities (if you are working with a document, you must make sure that the semantic representation of a chart aligns with the semantic representation of the text discussing the same chart)

Approaches to Deal With MultiModal Data:

- Embed all modalities into the same vector space
- Ground all modalities into one primary modality
- Have separate stores for different modalities

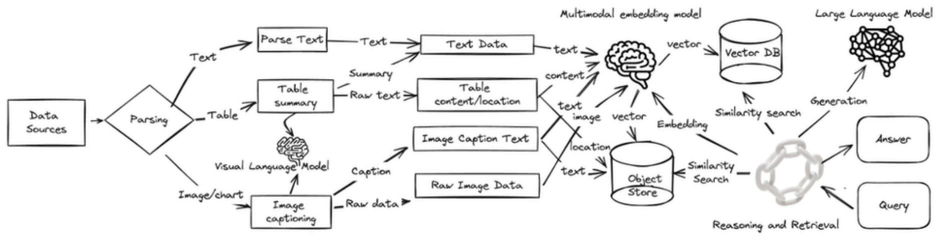
An approach to build Multimodal RAG

Link: Industry’s First Multimodal RAG on Dell PowerEdge XE9680 Server with AMD Instinct MI300X Accelerators | Dell Technologies Info Hub

- MilvusDB, an open-source vector database
- OpenAI Whisper, an automatic speech recognition system (ASR)
- CLIP, an image embedding model
- bge-large-en-v1.5, a text embedding model, which captures syntactic and semantic information from text data and encapsulates the information in numerical vectors
- Llama 3 with vLLM, with LlamaIndex as a RAG Framework
- Fast API, to interface with user interactions including video file uploads, model selection, and using the conversation console

Multimodal RAG Architecture

Create a multimodal assistant with advanced RAG and Amazon Bedrock | Amazon Web Services



Jina CLIP V1

Jina AI has improved on OpenAI CLIP’s performance by 165% in text-only retrieval, and 12% in image-to-image retrieval, with identical or mildly better performance in text-to-image and image-to-text tasks.

Open Source Embedding Models

- GTE-Base

General model for similarity search or downstream enrichments.

Used for general text blobs.

Limited to 512 tokens

Embedding Dimensions: 768

Model size: 219 MB

- GTE-Large

High quality general model for similarity search or downstream

Used for general text blobs

Limited to 512 tokens

Embedding Dimensions: 1024

Model Size: 670 MB

- GTE-Small

Good quality general model for faster similarity search or downstream

Used for general text blobs

Limited to 512 tokens

Embedding Dimensions: 384

Model Size: 67 MB

- E5-Small

A good small and fast general model for similarity search or downstream enrichments

Used for general text blobs

Limited to 512 tokens

Embedding Dimension: 384

Model size: 128 MB

- Multilingual

A general model to handle multilingual datasets

Used for general text blobs

Limited to 512 tokens

Embedding Dimension: 768

Model Size: 1.04 GB

- RoBERTa

A RoBERTa model train on data up to december 2022, for users who are familiar with BERT model family and want to use it in Graft

Used for general text blobs

Limited to 512 tokens

Embedding Dimensions: 768

Model Size: 476 MB

- MPNet V2

Mpnet model with Siamese architecture trained for text similarity

Used for similarity search for text

Limited to 512 tokens

Embedding Dimensions: 768

Model Size: 420 MB

- Scibert Science-Vocabulary Uncased

A BERT model pretrained on scientific text with specific science focused vocabulary

Used for scientific text blobs

Limited to 512 tokens

Embedding Dimensions: 768

Model Size: 442 MB

- Longformer Base 4096

A transformer model for long text, based on RoBERTa

Used for text up to 4096 tokens

Limited to 4096 tokens

Embedding Dimensions: 768

Model Size: 597 MB

- Distilbert Base Uncased

Relatively fast and small model with near performance to BERT

Used for general text blobs

Limited to 512 tokens

Embedding Dimensions: 768

Model Size: 268 MB

- Bert Base Uncased

The BERT Language model trained on English text via masked language modeling and next sentence prediction

Used for general text blobs

Limited to 512 tokens

Embedding Dimension: 768

Model Size 440 MB

- Multilingual BERT

A multilingual version of BERT trained on 102 languages

Used for scenarios where text of various languages exist

Limited to 512 tokens

Embedding Dimensions: 768

Model Size: 714 MB

- E5-Base

A good general model for similarity search and downstream enrichment.

Used for general text blobs

Limited to 512 tokens

Embedding Dimensions: 768

Model Size: 418 MB

- LED 16k

A transformer model for very long text, based on BART

Used for text up to 16384 tokens

Limitation: Compressing 16kish words into 768 dimensions will surely be lossy

Embedding Dimensions: 768

Model Size: 648 MB

- Voyage-Lite-02-Instruct

Instruction tuned model from first generation of the Voyage family

Used for instruction tuned for classification, clustering and sentence textual similarity tasks, which are the only recommended use cases.

Limited to small text embedding model from second generation of Voyage family

Embedding Dimension: 1024

Model Size: 1220 MB

- stella_en_1.5B_v5

Models is trained on Alibaba-NLP/gte-large-en-v1.5 and Alibaba-NLP/gte-Qwen2-1.5B-instruct

Model is finally trained by MRL and they have multiple dimensions: 512, 768, 1024, 2048, 4086, 6144 and 8192.

The higher the dimension, the better the performance.

Max tokens: 121072

Model Size (Million Parameters) 1543

- SFR-Embedding-2_R

Embedding Dimension: 4096

Max Tokens: 32768

Model Size (Million Parameters): 7111

- BGE-Large-en-v1.5

Supports more languages, longer texts and other retrieval methods.

Max tokens: 512

First embedding model that supports all 3 retrieval methods.

Embedding Dimensions: 1024

Model Size (Million Parameters): 335

- Jina-Embeddings-V2

Embedding Dimensions: 1536

Max Token: 8191

Model Size: 0.27 GB

Best used for legal document analysis, medical research, literary analysis, financial forecasting, conversational AI.

- BGE-M3

It's distinguished for its versatility in multi-functionality, multi-linguality and multi-granularity.

It can simultaneously perform the three common retrieval functionalities of embedding model: Dense retrieval, multi-vector retrieval, and sparse retrieval.

Multi-Linguality: It can support more than 100 working languages

Multi-granularity: It is able to process inputs of different granularities, spanning from short sentences to long documents of up to 8192 tokens.

Model Size: 2.2 GB

- GTE-Qwen2-7B-instruct

It ranks no.1 in both English and Chinese evaluations on the Massive text embedding benchmark.

GTE-Qwen2-7b-instruct is trained on model based on the Qwen2-7b LLM Model.

The model incorporates several advancements

Integration of bidirectional attention mechanisms, enriching its contextual understanding.

Instruction tuning, applied solely on the query side for streamlined efficiency

Comprehensive training across a vast, multilingual text corpus spanning diverse domains and scenarios.

Model Size: 7B

Embedding Dimension: 3584

Max Input Tokens: 32k

Best Strategies to Select an Embedding Model

- MTEB Leaderboard on Hugging Face is a good starting point to shortlist the top notch models. However, it is still not a 100 percent accurate comparison as these results may vary with your dataset.
- A model with higher NDCG is better at ranking relevant items higher in the list of retrieved results.
- Model size has a direct impact on latency. The latency performance trade off becomes especially important in a production setup.
- Max tokens supported by a model indicate the number of tokens that can be compressed into a single embedding. Generally a model with 512 tokens is more than enough
- Smaller embeddings offer faster inference and are more storage efficient, while higher dimensions can capture nuanced details and relationships in the data. So it will be a trade-off between capturing the complexity of the data and operational efficiency.

According to a research presented in this blog: https://towardsdatascience.com/openai-vs-open-source-multilingual-embedding-models-e5ccb7c90f05

BGE-M3 outperformed the open AI models in terms of almost everything.

Topic : Embeddings

COURSE : 1

Embedding Models: from Architecture to Implementation - DeepLearning.AI

Introduction to Embeddings:

- Embedding models create vectors that enable semantic or meaning-based retrieval systems.

Key Concepts:

1. **Embedding Vectors:**

- Represent words, phrases, sentences, or other entities as points in a vector space.
- Similar points in vector space have similar semantic meanings.

2. **Word Embeddings:**

- Word2Vec (Google Brain): Predicts a word based on context words.
- GloVe (Stanford University): Simplified mathematical approach, increases context window using recurrent neural networks and transformers.

3. **Transformers and BERT:**

- Transformers : Efficient feed-forward neural networks for sequential data.
- BERT : Deep transformer networks trained on fill-in-the-blank tasks for deep language understanding, enabling advanced NLP applications.

Applications of Embeddings:

- Text embeddings for words, phrases, sentences, images, videos, and audio clips.
- Used in LLMs, semantic search, product recommendations, anomaly detection, etc.

Retrieval Systems:

- **Cross Encoder:**
- Concatenates question and answer, computes relevance.
- Accurate but computationally intensive.

- **Sentence Embeddings:**
- Embeds text segments into vectors for fast similarity search.
- Less accurate than cross encoders but faster and scalable.

Lesson 1: Word and Sentence Embeddings

- **Vector Embeddings:**
- Map entities to vectors in space.
- Dense vector representations capture semantic meaning.
- **Applications:**
- Text, images, audio clips using models like CLIP (OpenAI).
- Semantic search, product recommendations, anomaly detection.

- **Sentence Embeddings:**

- Apply word embedding principles to sentences.
- Use in RAG (Retrieval-Augmented Generation) pipelines for query and response embeddings.

Lesson 2: Contextualized Token Embeddings

- **Problem with Word Embeddings:**
- Lack of context sensitivity.
- **Transformers:**
- Introduced in 2017, solve context problems.
- BERT model: Encoder-only transformer for contextualized embeddings.
- Pretrained on Masked Language Modeling (MLM) and Next Sentence Prediction (NSP).
- Effective in tasks like classification, NER, question answering.

Lesson 3: Sentence Embeddings Development

- **Tokenization:**
- Subword tokenizers (BPE, WordPiece, SentencePiece) used in transformers.
- **Initial Attempts:**
- Mean pooling of token embeddings or using CLS token failed.
- **Successful Approach:**
- Dual encoder architecture with contrastive loss.
- Trains separate encoders for questions and answers.
- Use in sentence similarity and RAG applications.

(Fine tuning)

https://github.com/meshalJcheema/Dual-encoder-Training Connect your Github account

Lesson 4: Building and Training Dual Encoders

- **Architecture:**
- Two independent BERT encoders for questions and answers.
- Uses CLS token embeddings, contrastive loss to optimize similarity.

- **Training Process:**
- Cross entropy loss to match diagonal similarity scores.
- Training loop iterates over batches and epochs to optimize encoders.
- **Practical Example:**
- Small model trained with PyTorch.
- Demonstrates the importance of adequate data and epochs for effective training.

Lesson 5: Using Embeddings in Production

- **RAG Pipeline:**
- Encode text chunks with answer encoder, store in vector database.
- Query encoder generates query embedding, retrieves matching chunks.
- Use ANN algorithms (HNSW, Annoy, FAZE) for efficient retrieval.
- **ANN Implementation:**
- Approximate nearest neighbor searches for fast and accurate retrieval.

Conclusions:

1. **Effectiveness of Embeddings:**

- Word embeddings (Word2Vec, GloVe) capture semantic meaning but lack context.
- Contextualized embeddings (BERT) address context but need sophisticated architectures for full sentences.

2. **Dual Encoder Advantage:**

- Separate encoders for queries and responses enhance relevance in RAG systems.
- Contrastive loss ensures semantic matching, making it suitable for real-world applications.

3. **Practical Implementation:**

- ANN algorithms are essential for scaling embedding-based retrieval.
- Two-stage retrieval (retrieve and re-rank) combines speed and accuracy, leveraging both sentence embeddings and cross encoders.

4. **Future Directions:**

- Embedding models are crucial for advanced NLP tasks, including hybrid and multimodal search systems.

What is a key characteristic of vector embeddings in natural language processing?

Points in vector space that are similar to each other have similar semantic meaning.

Which model developed by OpenAI aligns images and text in a shared representation space?

CLIP

What is a limitation of using a cross encoder in a RAG pipeline?

It is very slow and does not scale well.

What is the main problem with traditional word embedding models like Word2vec?

They do not understand the context of words in a sentence.

What are the two tasks BERT was pre-trained on?

Masked Language Modeling and Next Sentence Prediction

What does a cross-encoder do in the context of a retrieval pipeline?

Generates new sentences from a given text.

What is the main issue with using mean pooling for sentence embeddings?

It fails to integrate the context of the entire sentence.

What is the primary purpose of using a dual encoder architecture in sentence embedding models?

To separate the processing of questions and answers using two independent BERT encoders.

What role does contrastive loss play in the dual encoder architecture?

It ensures that similar pairs of data points are closer in the embedding space and dissimilar pairs are further apart.

What are the two types of encoders used in the dual encoder architecture for sentence embedding models?

Question Encoder and Answer Encoder

SQL Agents

Langchain SQL Agent

Advantages of the SQL Agent

- **Schema Awareness**: SQL Agents can interpret and respond to queries based on the database schema, allowing for more informed responses.
- **Error Recovery**: They can handle errors gracefully by regenerating queries after catching exceptions, ensuring smooth operations.
- **Multi-Query Handling**: SQL Agents can manage questions that require multiple dependent queries, streamlining the querying process.
- **Token Efficiency**: By focusing only on relevant tables, they save tokens, making them cost-effective.

How the LangChain SQL Agent Works

1. **Agent and Toolkit Setup**: The LangChain SQL agent is created using the `create_sql_agent` function, which connects a language model (such as OpenAI's GPT models) with a `SQLDatabaseToolkit`. This toolkit is responsible for managing the interaction between the LLM and the database. The toolkit itself is configured with the specific database you want to query, which could be an SQLite, PostgreSQL, or any other SQL-compatible database.
2. **Understanding the Database Schema**: When the SQL agent is initialized, it performs several operations to understand the structure of the database:
   - **Listing Tables**: The agent first lists the tables available in the database using commands like `sql_db_list_tables`.
   - **Querying Schema**: It then queries the schema of the relevant tables to understand their structure. For example, it fetches the column names and data types by running a command like `sql_db_schema` on a specific table. This allows the agent to understand what kind of data each table holds and how it can be queried.
3. **Generating SQL Queries**: Once the schema is understood, the agent uses the LLM to translate natural language queries into SQL queries. It does this by leveraging the schema information and example rows from the database. For example, if a user asks for the average temperature at a specific station within a date range, the LLM constructs the corresponding SQL query that can retrieve this data.
4. **Executing SQL Queries**: After constructing the SQL query, the agent executes it against the database. The result is then processed and formatted by the LLM to provide a user-friendly response. The agent also supports more advanced use cases, like handling complex joins, aggregations, or filtering based on conditions specified in natural language.
5. **Applications**: The LangChain SQL agent is highly versatile and can be used in various scenarios, including:
   - **Data Analysis**: Users can ask questions about their data without writing SQL, which is especially useful for quick analyses.
   - **Database Management**: Beyond data retrieval, the SQL Agent can assist in database management tasks, such as schema exploration and data integrity checks, by interpreting the database's structure and content through LLMs.
   - **Automated Reporting**: The agent can generate reports by querying and summarizing database content automatically.
   - **Educational Tool**: It helps users learn SQL by showing how natural language queries are translated into SQL.

Key Features:

- **Dynamic Query Generation**: The agent dynamically generates SQL queries based on natural language inputs, allowing for flexible and intuitive data retrieval.
- **Integration with Various Databases**: It supports a wide range of SQL databases, including but not limited to MySQL, PostgreSQL, and SQLite, through the use of the SQLDatabaseToolkit.

- **Verbose Mode for Debugging**: A verbose mode is available, providing detailed insights into the agent's decision-making process and the SQL queries being executed.
- **Schema Understanding**: Interprets database schema to identify relevant data points for the query.
- **Conversation Context**: Maintains context over a conversation, allowing for follow-up questions without repeating context.

References

- LangChain's official blog
- Restack's documentation (LangChain Blog) (AI Product Builder).

Llama Index:

How the Llama Index SQL Agent works:

1. Database Schema Identification

- **Table Schema Indexing:** LlamaIndex begins by creating a structured index from the schema of the SQL database. This involves defining the structure of the SQL tables within LlamaIndex, specifying the columns, data types, and relationships between tables. This index acts as a bridge between the database's raw structure and the queries posed by users in natural language.

2. Natural Language Query Processing

- **Text-to-SQL Translation:** LlamaIndex uses its `RetrievalQueryEngine` to convert natural language queries into SQL commands. This engine interprets the user's query, understands the intent, and generates the appropriate SQL code to fetch the required data from the database.
- **Context Injection:** LlamaIndex can inject context into the SQL queries. This context can be manually added or derived from unstructured data, providing additional information that might be necessary for accurate query execution. This is particularly useful in cases where the query is ambiguous or requires more detailed understanding of the data.

3. Query Execution and Data Retrieval

- **Executing Queries:** Once the SQL command is generated, it is executed against the database. The results are then processed and returned to the user in a format that is easy to understand, often as a natural language response or a structured data table.
- **Advanced Retrieval Techniques:** LlamaIndex supports augmented Text-to-SQL capabilities, which means it can handle more complex queries and data analysis tasks. This includes dealing with multi-step queries, integrating data from different sources, and providing context-aware responses.

References:

- Arize AI
- Llama Index

RAG Over Confluence

Goal: Extract data from confluence pages and apply RAG pipeline to retrieve accurate results

Steps:

1. Get confluence api key, space key, url, username to access contents of the confluence space.
2. Select a data loader to get the content from the confluence pages
3. Store data in a db vector and use it in RAG pipeline.
4. Evaluate responses using an evaluator to compare accuracy.

- First approach (Langchain Data loader)
    - Extracted data from confluence pages only for texts. Didn’t consider tables.
    - Data Loader: Langchain
    - LLM: OpenAI (gpt 3.5 turbo)
    - Embedding Model: OpenAIEmbeddings
    - Evaluator: RAGAS
    - Result analysis: Results were fine but not so good. It worked good with fewer files but when the files were more then it didn’t work very effectively.
- Second approach (LlamaIndex Data loader)
    - Extracted data from confluence pages for tables and texts.
    - Data Loader: LlamaIndex
    - LLM: OpenAI (gpt 3.5 turbo), LLama3 7b
    - Embedding Model: OpenAIEmbeddings, `sentence-transformers/all-mpnet-base-v2`
    - Evaluator: RAGAS
    - Result analysis: Results were good for plain text but retrieval from tables was a failure
- Third approach (Unstructured Data with pdf parser & Langchain)
    - Extracted data from confluence pages.
    - Converted those pages to pdf files.
    - Data Loader: Langchain
    - LLM: OpenAI (gpt 3.5 turbo)
    - Embedding Model: OpenAIEmbeddings
    - Evaluator: RAGAS
    - Result analysis: Results were not constant. Sometimes it worked, sometimes it didn’t. Retrieval was correct as returned sources were correct. There was a problem in the answer generation.
- Fourth approach (Unstructured Data with pdf parser & Langchain)
    - Extracted data from confluence pages.
    - Converted those pages to pdf files.
    - Data Loader: Langchain
    - LLM: OpenAI (gpt 3.5 turbo), LLama3 7b
    - Embedding Model: OpenAIEmbeddings, `sentence-transformers/all-mpnet-base-v2`
    - Splitter: Recursive Splitter
    - Evaluator: RAGAS
    - Result analysis: Results were not so accurate. For some queries, it worked fine but for others, not an acceptable performance.
- Fifth approach (Unstructured Data with LLamaIndex)

    - Extracted data from confluence pages.
    - Converted those pages to pdf files.
    - Data Loader: LLamaIndex
    - LLM: OpenAI (gpt 3.5 turbo), LLama3 7b
    - Embedding Model: OpenAIEmbeddings, `sentence-transformers/all-mpnet-base-v2`
    - Parser: Lama Parser
    - Reranker: BGE-Reranker
    - Evaluator: RAGAS
    - Result analysis: Results were fine. Was returning good results for more of the queries but for some, it was hallucinating.
- Sixth approach (Unstructured Data with LLamaIndex)
    - Extracted data from confluence pages.
    - Converted those pages to pdf files.
    - Data Loader: LLamaIndex
    - Parser: Lama Parser
    - Reranker: BGE-Reranker
    - Evaluator: RAGAS
    - Result analysis: Results were accurate 99% of the times. The summaries retrieved were accurate however, if I add an llm, it started to hallucinate at certain queries.

Fine Tuning Embedding's

EXPERIMENT : 1

Initial Dataset : 11 PDF documents

Training Dataset : 7 PDFs

Testing Dataset : 4 PDFs

Queries / Corpus Set for Training : 343

(generated by splitting text/chunks from pdfs, sending them to gpt35 to get Queries for that specific chunk)

Queries / Corpus Set for Testing : 520

(generated by splitting text/chunks from pdfs, sending them to gpt35 to get Queries for that specific chunk)

Number of Top k retrieved context = 5

Comparisons

Embedding	Output Dimension	Training Parameters	Hit Rate
`text-embedding-ada-002` (openAI)	1,536	350 million	92.6 %
`BAAI/bge-small-en`	384	20-100 million	88.8%
`Fine tuned BAAI/bge-small-en`	384	20-100 million	92.2%
`BAAI/bge-m3`	1,024	up to 500 million	94.2%
`Fine tuned BAAI/bge-m3`	**1,024**	**up to 500 million**	**99.4%**

EXPERIMENT : 2

Initial Dataset : 132 PDF documents (Extreme AI Expert) dataset

Training Dataset : 90 PDFs

Testing Dataset : 42 PDFs

Queries / Corpus Set for Training : 5522

Queries / Corpus Set for Testing : 5970

Number of Top k retrieved context = 10

Comparisons

Embedding	Output Dimension	Training Parameters	Hit Rate

`text-embedding-ada-002 (openAI)`	1,536	350 million	62.3 %
`BAAI/bge-small-en`	384	20-100 million	44.5%
`Fine tuned BAAI/bge-small-en`	384	20-100 million	69%
`BAAI/bge-m3`	1,024	up to 500 million	58%
`Fine tuned BAAI/bge-m3`	**1,024**	**up to 500 million**	-

CONCLUSION SO FAR:

1. Fine tuning does increase the retrieval hit rate

2. Models with more training parameters offer better hit rate

3. Dimensions in a vector space shall not be either very high subjected to over fitting nor very low subjected to under fitting

4. Models performances depends upon data semantics

5. BGE - small is better than OpenAI embedding's in both experiments leveraging the importance of fine tuning

NEXT

1. Try to fine tune BGE-M3

2. Fine tune an adapter(k-NN layers over any embeddings model)

3. Customize OpenAI embeddings

LLM Testing

Confluence Pages Exploration

CodeHawk

Summary

This is a summary of what has been understood by reviewing the CodeHawk documentation on Genie Confluence space. The major focus of this summary will be on the Generative AI components of CodeHawk.

CodeHawk utilizes large language models to automate code reviews when a PR is created. It gives suggestions for improvements, generates explanation for components, and also facilitates discussion threads. In the interactive clarification feature, developer can tag @CodeHawk to get additional explanations from the LLM in the PR comments. Each new commit in a pull request triggers incremental reviews, forming sort of a feedback loop (if the review label is added). LLM takes in all the context including the repo and file level context. Context window size was set to 10 so it takes 10 lines before changed code and 10 lines after changed code. ReAct agent is also integrated into the system; when the first LLM produces review, everything from context to review is fed into another LLM (GPT-4) to assess if review can be improved.

Multiple models like Llama2-70b, Codegen2-7B, and Mistral 7b were tested for code review quality and based on performance Mistral 7b Instruct was identified as having a stronger understanding of the code. Codegen2-7B focused more on generating code rather than analyzing it. Llama3-70b was used to evaluate all other models; Llama2 models received a lower score compared to all others; Mistral 7b Instruct finetuned received a score of 8 out of 10 and Phi 3 received a score of 9.

Final model used is Llama3-70B; Locust was also used to test response time and concurrency.

Data Points for Testing

1. **Accuracy of PR review**
   a. **objective:** Validate that the LLM accurately generates code review according to the context.
   b. **scope:** LLM should be able to detect syntax errors, logical errors, and violations of coding conventions in regards to the changed code and give feedback related to surrounding code too.
   c. **approach:** Provide code snippets in various programming languages with syntax and logical errors while varying the context window.
   d. **expected outcome:** LLM should give accurate, relevant, and comprehensive feedback in regards to the whole context, not just the code lines. The suggestions should be actionable.
   e. **metrics:** accuracy, helpfulness and relevance
2. **AI powered explanation in threads**
   a. **objective:** Validate that the LLM provides clear, detailed, and relevant to context explanations when developer tags CodeHawk in comment thread.
   b. **scope:** LLM should be able to explain the reasoning behind its feedback on a code review and reference the previous comments and discussions in a thread to give coherent and relevant explanations.
   c. **approach:** Simulate interactions where further clarification is requested on initial code review by tagging CodeHawk; give queries that require in-depth explanations to see depth of LLM response and track if LLM can reference ongoing discussion in responses.
   d. **expected outcome:** LLM gives clear, concise, and relevant explanations that address the query.
   e. **metrics:** relevance, clarity
3. **Incremental review functionality**
   a. **objective:** Validate that the LLM provides feedback only on newly added or changed code and avoids redundant reviews of code previously analyzed.
   b. **scope:** LLM should focus on incremental change that comes with each commit to a previously analyzed.
   c. **scope:** LLM should focus on incremental change that comes with each commit to a pull request and should only review new code, skipping previously reviewed sections.

   d. **approach:** Simulate situations with multiple commits in a pull request to ensure LLM gives feedback only on latest commit; vary scope with minor fixes or larger additions.
   e. **expected outcome:** LLM can identify and give feedback on new changes without duplicating feedback from prior commits, in context with the full code.
   f. **metrics:** accuracy, consistency
4. **ReAct agent refining initial reviews**
   a. **objective:** Assess how effective the ReAct agent is in refining the LLM reviews by giving additional improvements, insights rather than just confirming the review.
   b. **scope:** ReAct agent should identify shortcomings of initial review and give better suggestions that add value to the original review.
   c. **approach:** Give edge cases where original review misses complex logical errors, or improvements and observe how ReAct agent introduces improvements.
   d. **expected outcome:** ReAct agent gives valuable improvements to original review in terms of specific and actionable feedback and not just restate the original review.
   e. **metrics:** relevance, quality

Rag Over Confluence

Summary

This is a summary of what has been understood by reviewing the Rag over Confluence documentation on Genie Confluence space. The major focus of this summary will be on the generative AI components of the project.

The project's aim is to extract data from Confluence pages and apply a RAG pipeline to retrieve accurate results. Different approaches were utilized using different data loaders like Langchain and LlamaIndex. The focus was on extracting text and table data from Confluence pages. Various LLMS like GPT 3.5 and Llama3 7B were used along with different embedding models like OpenAI embeddings and sentence transformers. The evaluator used was RAGAS. The initial attempts contained a lot of inconsistent results especially when considering a large amount of data and retrieving information from tables. The following iterations had an improved retrieval accuracy by using parsers like LlamaParser and rerankers like BGE-Reranker. The best approach identified so far uses LlamaIndex and LlamaParser. This had a high accuracy in terms of retrieval but when an LLM was added, some hallucination issues were present.

Data points for testing

1. **Evaluation of LLMs answer generation**
   a. **objective:** Evaluate how well the LLM generates contextually accurate and useful responses based on the retrieved document chunks.
   b. **scope:** The LLM should be able to generate meaningful answers based on the chunks retrieved. Primary focus is on whether LLM is interpreting the data.
   c. **approach:** Give the LLM queries that need understanding and retrieval of both text and table data; judge the response's accuracy and quality and detect hallucinations.
   d. **expected outcome:** Accurate responses based on retrieved documents generated.
   e. **metrics:** accuracy, hallucination rate
2. **Consistency of query responses**
   a. **objective:** Ensure LLM consistently gives accurate answers when the same query is submitted over time.
   b. **scope:** LLM should be able to generate accurate answers consistently even with updates to dataset.
   c. **approach:** Give the same query at different times to the LLM and compare the responses for consistency; modify data and try again.
   d. **expected outcome:** Consistent responses for the same query unless the change is due to a valid reason.
   e. **metrics:** consistency, accuracy

Run Existing Github Repositories

CodeHawk

- The .yml file and gitlab access were provided by Arshik.
- Imported github repo of langchain assignment 1 in gitlab to run codehawk on it.
- Created a small typo in code and set up the pipeline and created merge request but pipeline failed due to 404 project not found error. Contacted Arshik about it.
- Issue was resolved by hardcoding own gitlab access token into the .yml script.

Rag over Confluence

- Access to github repo was provided by Mahnoor.
- The various api keys to be used were also provided by Mahnoor.
- First VS code took 55 minutes to execute nltk.download(all) because of internet issues so I had to interrupt that. Then it took 90 mins and I had to interrupt it again. After that I only downloaded some modules I believe were needed.
- Fast Embed package not installing with error ‘ERROR: ERROR: Failed to build installable wheels for some pyproject.toml based projects (PyStemmer).’ Had to install Microsoft C++ Build tools.

Test Cases for PR Review

Test Cases for Advance RAG

Manual Testing for PR Review