

Genie Research and POCs Home	3
Untitled Smart Link 1	4
Advance RAG	5
Comparing LLMS for Rag Over Code	6
Advanced RAG Taxonomy	8
AdvanceRAG Tools Comparison	9
Advance RAG : Query Optimization	13
Advance RAG : Document Optimization	15
Advance RAG : Embedding Models	17
Advance RAG : Context Expansion	19
Advance RAG : Context Rerankers	21
Testing Frameworks	24
RAG over different types of Data	25
Structured Planning Agent Llama-index	27
Explore Evaluation Metrics	29
Evaluation Metrics	31
Q/A with RetrieverRouter engine from llama-index	33
Recursive Retrieval Techniques	34
Exploring JSON query engines of llama-index	36
Capture Explanations and Compare Our RAG Mindmap	41
Exploring text-to-SQL Query engine of llama-index	43
Q/A over unstructured vs structured data	44
Q/A with ReAct agent for routing	46
RetrieverRouterQueryEngine vs StructuredPlanningAgent vs ReAct for query routing	50
Mini-Assistant : Combining Q/A and function calling tools with ReAct on top for agentic reasoning	51
Code Review	54
Arshik's Day-by-Day Summary of the Genie Learning Path	55
Potential Features for CodeHawk Emumba	57
CodeHawk GitHub Implementation	58
CodeHawk GitLab Implementation	64
Running CodeHawk Scripts with Docker in GitHub Actions	65
Interactive Clarification (Feature 1)	68
Pull Request Summaries (Feature 2)	71
Integrating with JIRA Issue Tickets (Feature 3)	72
Comparison of PR Reviewer with other similar tools	73
Development Automation PoC	75
Self Hosting LLM	77
Fine Tuning Techniques-LLM	81
Fine tuning LLMs using Lora	84
Comparison of Codegen2-7B and Mistral 7b for Code Review Tasks	102
List Down the Resources Required for the Deployment of a Language Model on AWS EC2	105
Control RAG Output	107
Retrievers Comparsion	109
Context Window Solution	112
Stress Testing EC2 Endpoint	114

Literature Review of RAG/LLM Based Log Analysis or Incident Management or Root Cause Analysis.....	116
RAG Failure Causes.....	124
GraphRAG.....	127
Multimodal RAG & Open Source Embedding Models.....	129
Topic : Embeddings.....	137
SQL Agents.....	141
RAG Over Confluence.....	143
Fine Tuning Embedding's.....	145
LLM Testing.....	148
Confluence Pages Exploration.....	149
Run Existing Github Repositories.....	151
Test Cases for PR Review.....	152
Test Cases for Advance RAG.....	153
Manual Testing for PR Review.....	154

Genie Research and POCs Home

Welcome to your new space!

Spaces help your team structure, organize, and share work, so every team member has visibility into institutional knowledge and access to the information they need to do their best work.

Structuring Research and Development Tasks

We will broadly have two types of tasks in this project, either research/learning related or development (proof of concept). We will follow the practices below:

- A completion criteria should be specified with each task
- Due dates are optional but should always be specified with time-bound tasks
- Before marking a task as complete, it is compulsory that you add a brief conclusion/findings in a Confluence page linked to the task description. This applies to both research and development tasks. e.g. if you were tasked with research on a new idea, summarise your understanding. If you tried something new in code, summarise your conclusion and any instructions, set up etc for reproducing your findings.
- All notebooks should be maintained in the Genie Github repo and must be committed to the repo before a task/subtask is marked completed. Also link the notebook in the description of the task or the associated Confluence page.
-

Untitled Smart Link 1

Advance RAG

Rag over pdf

- Number of pdfs - 3 (all from different domains)
- Queries - 50
- Dataset fields - (Queries, Contexts, Ground Truth)
- GitHub - Rutuja161/Pdf-based-question-answering: This is the research project for my masters in Data analytics. This aims to build a smart question answering system for PDF files using BERT - based transformers.
- Evaluators used : 2

Rag over Code POC

- Number of repos - 1
- Queries - 10
- GitHub - emumba-com/emumba-nx-boilerplate-plugin

GitHub - https://github.com/meshalJcheema/Advance-Rag-Mindmap-usecases Connect your Github account

- Evaluators used : 5

Evaluators

- Following are the two metrics used from evaluators
- **Response Matching/Answer Relevance** : Measures the percentage of responses that exactly match the expected answer. (uses ground truth)
- **Context Relevance** : measures if the retrieved context has enough information to answer the question being asked. how good is the context retrieved

Comparing LLMS for Rag Over Code

Extract content from Github Folder and Split using Langchain

This function does the following:

- Defines a Document class to store file content and metadata.
- Reads files from a specified path and creates Document objects for each file.
- Splits the original documents into smaller chunks using the text splitter, creating a new list of documents.

PIPELINE - 1

GENIE-3 & GENIE-4 & GENIE-9 : Pipeline With Different Bedrock LLMs and Embeddings

Following Pairs were made for analysis:

1. Model Claude-V2, Embeddings - model_id="amazon.titan-embed-text-v1"
 2. Model llama 2 70b, Embeddings - model_id="amazon.titan-embed-text-v1"
 3. Model llama 2 70b, Embeddings - model_id="cohere.embed-english-v3"
 4. Model llama 2 70b, Embeddings - model_id="text-embedding-ada-002"
- The following pipeline uses FAISS as its vectorDB.

GENIE-5 : Evaluation of pipeline using LLM

- The llm used is the claude V2 from bedrock service.
- The process automates the evaluation of answer quality, using a language model to assign relevance scores to pairs of original and candidate answers based on their context and query.

PIPELINE - 2

GENIE-6 : RAPTOR OVER RAG

- The following pipeline uses openai embeddings model and bedrock model claude as its llm
- Imports OpenAIEmbeddings and ChatOpenAI from langchain_openai, and various other classes and functions from langchain for embedding and model interaction purposes.
- Introduces a recursive clustering and summarization function that aims to embed texts, perform clustering at multiple levels, and summarize the texts within each cluster.
- The summarization part is intended to generate a summary for each cluster of texts by utilizing a language model (presumably Claude-v2), following a specified prompt template and output parsing.
- Contains a loop at the end to process a list of queries, generate responses using a chain that involves document retrieval, prompting, and the Claude-v2 model for summarization, and then saves each response to a separate JSON file.

PIPELINE - 3

GENIE-8 : OLLAMA EVALUATION

- The llm is used is llama2 and embeddings used are GPT4allembbeddings.

GENIE-7 : RAG EVALUATION

- The llm used is Claude - V2 from bedrock service, OpenAI embeddings and Weaviate vectorDB for embeddings manipulation.
- Assembles the queries, answers, contexts, and ground truth answers into a dataset.
- Evaluates the dataset using several metrics (context precision, context recall, faithfulness, and answer relevancy) to assess the quality of the generated answers.
- Converts the evaluation results into a pandas DataFrame and prints it.
- Saves the evaluation results to a CSV file named "ragas_results.csv".
- Overall, this pipeline demonstrates a workflow for answering queries with context from a vector database, generating answers with a language model, and evaluating the quality of those answers against ground truth data.

Advanced RAG Taxonomy

The task is designed to develop a comprehensive taxonomy for retrieval augmented generation models. The main goal of the taxonomy is to provide a structured framework by categorising the tools and techniques used in different components of RAG pipeline.

Link to Mind Map: <https://mm.tt/app/map/3231556693?t=U72qrjV59q>

Approach:

A comprehensive literature review was conducted to understand the RAGs and its pipeline. This involved studying various survey papers, article and frameworks.

Based on the the gained insight, different components were identified and characterised.

The identified components were carefully fitted into the pipeline of the advanced rag model.

The Rag pipeline is mainly divided into 6 components:

1- Pre-Retrieval: It is used before the retrieval process. Main purpose is to pre-process the user query and documents.

2- Retrieval: This component is used in the overall retrieval process that includes embedding models and retrieval algorithms.

3- Generation: It comprises of the post-retrieval part. That further involves relevance enhancement and LLM- FINE TUNING. Relevance enhancement involves context expansion, response re-ranking and embedding adapter.

4- Augmentation: It mainly comprises of 3 different retrievals that helps in the augmentation process.

5- RAG Evaluation: Various evaluations methods are categorised along with their implementations.

6- Tools: Famous tools that are being used for RAG implementation are classified along with their implementations.

Information Sources Categorised by Colors:

- Green: Git hub Repository
- Blue: Research Paper
- Yellow: Official Library
- Pink: Blog/ Website/ Colab notebook

AdvanceRAG Tools Comparison

Here's a comparative tables based on comparison for the following tools

1. LlamaIndex

2. LangChain

3. Haystack

4. Hugging Face

Definition Analysis between Tools

Category	LlamaIndex	LangChain	Haystack	Hugging Face
Supported Languages	Python TypeScript	Python JavaScript	Python	Python
Main Focus	<ul style="list-style-type: none">Indexing and efficient retrieval	<ul style="list-style-type: none">General-purpose LLM applicationsRAG approachcontext-aware apps	<ul style="list-style-type: none">Semantic search and Q&A systems	<ul style="list-style-type: none">Pre-trained models, transfer learningwide range of NLP tasks
Key Features	<ul style="list-style-type: none">Integration with diverse data sources (160+)list indexcustomizable workflows	<ul style="list-style-type: none">Modular designflexible integrationdevelopmentdeploymentobservabilityaccess to various LLMs	<ul style="list-style-type: none">Vendor neutralityinteroperabilitycustomizabilityextensibilityspecialized components	<ul style="list-style-type: none">Pre-trained modelsextensive documentationlarge community easy transfer learningcustomization
Pros	<ul style="list-style-type: none">Handles wide range of data formatsAdvanced indexing techniquesCustomizable RAG process	<ul style="list-style-type: none">Modular architectureExtensive ecosystemAccess to various LLM providersEase of use	<ul style="list-style-type: none">Designed for search and Q&ASpecialized componentsIntegrated workflow	<ul style="list-style-type: none">Wide range of state-of-the-art modelUser-friendly APILarge communityCustomizable

Cons	<ul style="list-style-type: none"> Limited to data retrieval, less suitable for complex NLP tasks 	<ul style="list-style-type: none"> Generalized, not specialized- Performance may vary based on use case 	<ul style="list-style-type: none"> Narrower scope Primarily focused on document understanding and retrieval tasks 	<ul style="list-style-type: none"> Not specifically focused on search or retrieval systems
Use Cases	<ul style="list-style-type: none"> Q&A chatbots Document understanding and extraction Autonomous agents 	<ul style="list-style-type: none"> Prototyping and experimentation Enterprise-level applications 	<ul style="list-style-type: none"> Enterprise search and QA Specialized knowledge management 	<ul style="list-style-type: none"> Text classification Question answering Language translation
Vendor Integration	Customizable with different retrieval strategies	Integrates with multiple LLM providers like OpenAI, Cohere, Hugging Face Hub	Integration with various model providers and databases	Provides a wide range of pre-trained models from different providers
Community and Ecosystem	Growing community, lightweight interface	Large and active community, well-documented examples	Well-documented API, growing community	Extensive community and resources, active user base
Performance Optimization	Focused on efficient indexing and retrieval	Flexible but may not match performance of tailored solutions for specific tasks	Optimized for search and Q&A, better performance in semantic search	Optimized for various NLP tasks, transfer learning capabilities
Example Libraries/Tools		LangSmith, LangServe		Transformers

Advance Rag Use-Cases comparison

Feature	LlamaIndex	LangChain	Haystack	Hugging Face
Query Optimisation				
a) Multi-query expansion	✓	✓	✗	✗
b) Sub-query expansion	✓	✓	✗	✗
c) Hyde-adapter for embeddings	✓	✓	✓	✗
d) Rewrite queries	✗	✓	✗	✓
Documents Optimisation				

a) Chunking strategies	✓	✓	✗	✗
b) Metadata attachment	✓	✓	✗	✗
c) Hierarchical index structure	✓	✗	✗	✗
d) Knowledge graph index	✓	✗	✓	✗

Embeddings Model				
a) Ollama	✓	✓	✓	✓
b) Cohere	✓	✓	✓	✓
c) Jina	✓	✓	✓	✓
d) Voyage	✓	✓	✓	✓
e) OpenAI	✓	✓	✓	✓
f) LLM Embedder	✗	✗	✗	✓
g) BGE Embeddings	✗	✗	✗	✓

Context Expansion				
a) Sentence window retrieval	✓	✗	✗	✗
b) Recursive retrieval	✓	✗	✗	✗
c) Auto merging	✓	✗	✗	✗
Context Rerankers				
a) Cohere reranker	✓	✓	✓	✗
b) BGE reranker	✗	✗	✓	✓
c) Rank fusion	✓	✓	✓	
d) ColBERT reranker	✓	✓	✓	✓
e) Long context reorder	✓	✓	✗	✗
f) Jina reranker	✓	✓	✓	✓

g) RankGPT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
------------	-------------------------------------	-------------------------------------	-------------------------------------	-------------------------------------

Tools and Query Router Engines

Llamaindex

Llamaindex offers following query routing techniques

1. Pydantic Single Selector
2. LLM Single Selector
3. Pydantic Multi Selector
4. Query engine tools (define description in tools definition)

Langchain

Langchain offers following query routing techniques

1. Routing with function calling models
2. Routing to multiple indexes
3. Dynamic Routing based on custom logic related to what is your input

Haystack

Haystack offers following query routing techniques

1. Classifying keywords, questions and statement queries
2. Zero shot classification
3. Routing based on content type and metadata field
4. Conditional routers
5. Filetype routers
6. Text Language router
7. Transformerszeroshottextrouters

Hugging Face

Hugging face offers following query routing techniques

1. Hybrid LLM: Cost-Efficient and Quality-Aware Query Routing (paper)
2. Decomposing Complex Queries for Tip-of-the-tongue Retrieval (similar to subquery idea)

Advance RAG : Query Optimization

DATA : PDF documents, link mentioned on main page

Total queries : 50

Evaluation Tools are Uptrain & Tonic Validate

Evaluation Techniques are Baseline Experiment, Multiquery, Subquery, HYDE, Multi step, Step back, Query Rewriting.

Metrics for evaluation are Answer Similarity, Context Precision, Context Similarity.

Answer Similarity	Baseline Experiment	Multiquer y	Subquery	HYDE	Multi step	Step back	Query Rewriting
Uptrain	25%	28%	15%	48%	21%	27%	39%
Tonic Validate	44%	44%	30%	64%	48%	44%	50%

Context Precision	Baseline Experiment	Multiquer y	Subquery	HYDE	Multi step	Step back	Query Rewriting
Uptrain	45%	51%	22%	100%	44%	43%	49%
Tonic Validate	40%	64%	29%	80%	65%	63%	53%

DATA : Code files from a GitHub repository, link mentioned on main page

Total queries : 10

Context similarity	Multiquery	Subqueries	HYDE	Multi step	Step back prompting
Ragas	31%	82%	91%	91%	90%
Tonic Validate	1.5/5	3.2/5	4.4/5	1.2/5	1/5
DeepEval	10%	40%	85%	10%	45%
UpTrain	14%	34%	57%	23%	21%
Trulens	41%	56%	64%	76%	76%
Falcon-evaluate	25%	50%	96%	40%	33%

Conclusion

- HYDE performs best! - Expands search space, reduces hallucinations, we perform answer to answer embedding vector stores similarity that is resulting in better results.
- All other techniques,
- They highly depend on type of query user is going for.
- These techniques result in a lot of hallucinations.
- Therefore, on fairly decent queries present in our dataset, their scores lie near our baseline experiment except Hyde and query rewriting.

Advance RAG : Document Optimization

Experiment Results

DATA : PDF documents, link mentioned on main page

Total queries : 50

Context Precision	Baseline Experiment	Semantic Chunking	Tik token Chunking	Character text Chunking
Up train	45%	67%	69%	81%
Tonic Validate	40%	74%	70%	74%

Answer Similarity	Baseline Experiment	Metadata Attachment
Up train	25%	52%
Tonic Validate	44%	56%

Context Precision	Baseline Experiment	Metadata Attachment
Up train	45%	73%
Tonic Validate	40%	75%

Answer Similarity	Baseline Experiment	Hierarchical Indexing
Up train	25%	56%
Tonic Validate	44%	68%

Context Precision	Baseline Experiment	Hierarchical Indexing
Up train	45%	80%
Tonic Validate	40%	75%

DATA : Code files from a GitHub repository, link mentioned on main page

Total queries : 10

Context similarity	Recursive Chunking	Meta Data Inclusion	Semantic Chunking	Character Text splitting	Tik Token Based Splitting	Hierarchical Indexing

Ragas	80%	91%	77%	88%	71.08%	82%
Tonic Validate	2/5	1.9/5	1/5	2.1/5	0.9/5	4.4/5
Deep eval	35%	30%	49%	50%	40%	54%
Up train	26%	26%	22%	40.8%	16%	67%
Trulens	70%	72%	74%	75.5%	76%	76%
Falcon-evaluate	50%	46%	45%	59%	59%	96%

Conclusion

- There are multiple chunking techniques out there, depending upon use cases, code languages, extensions etc, we tried for our experiment the generic ones, all performed better, then baseline experiment, but **Character text splitting got maximum scores!**
- **Metadata integration** with baseline experiment boosted up the accuracy scores
- **Hierarchical indexing** dynamic parsing and creation of multiple layers of indices, not only gives you good results but it also has an optimized Response Time.
- Include metadata, necessary for scenarios where we have domain specific queries. Even include metadata with no domain specific, its good to have it
- The hierarchical and KI graphs can't be compared to chunking, they have different retrievers, nodes are there which are stored processed differently. Their main Plus seems to be their very fast storage and retrieval because of the way their structures are build but at the same time, they are much more complex to be understand-ed and integrated to our RAG pipeline

Advance RAG : Embedding Models

Variants and Embedding Size

< TO BE INCLUDED > SOON

Experiment Results

DATA : PDF documents, link mentioned on main page

Total queries : 50

Answer Similarity	Baseline Experiment (ollama)	OpenAI	JinaAI	BGE3	Cohere	Voyage AI	LLM embedder	Bedrock
Uptrain	25%	36%	38%	50%	37%	34%	28%	24%
Tonic Validate	44%	60%	54%	60%	53%	53%	48%	50%

Context Precision	Baseline Experiment (ollama)	OpenAI	JinaAI	BGE3	Cohere	Voyage AI	LLM embedder	Bedrock
Up train	45%	88%	54%	86%	52%	66%	52%	45%
Tonic Validate	40%	76%	69%	79%	74%	67%	55%	61%

DATA : Code files from a GitHub repository, link mentioned on main page

Total queries : 10

Context Similarity	Openai	Bge - m3	llm-embedder	Cohere	all-minilm	Mxbai-embedlarge	Voyage	Jina Ai
Ragas	91%	82.5%	80	77%	78%	79%	75%	76%
Tonic Validate	1.9/5	3.4/5	3/5	1.1/5	2.5%	3.1%	2.8/5	2.9/5
Deepeval	40%	33%	38	44%	36%	34%	39%	37%
Uptrain	33%	33%	35	24%	30%	32%	31%	32%
Trulens	75%	76.22%	76.22%	76%	76.22%	76.22%	76.22%	76.22%
Falcon-evaluate	55%	46%	48	33%	50%	47%	52%	49%

Conclusion

- All performed better than baseline experiment that uses minilm ollama embeddings except bedrock embeddings they were nearly equal to minilm ones.
- BGE-m3 embeddings got maximum scores — (multilingual, trained over short, long texts texts, 568 million parameters!)
- Between all others there does not lie much difference.
- Try them with a reranker for boosted results?
- BGE-m3, LLM embedders can't be integrated into our original pipeline, i made their own RAG pipeline and they need a GPU units, processing on CPU is very slow, in terms of model loading and initialization. Embedding formation time of T4-GPU is given above.

Advance RAG : Context Expansion

Following techniques are implemented, there is a general overview, how these techniques work.

Technique	Description	RAG use case
Sentence Window Retrieval	<p>Segmenting the corpus into overlapping sentence windows and scoring based on query relevance.</p> <p>Scoring each window based on its relevance to the input query.</p> <p>Providing highest scoring windows as context</p>	<p>Suitable for use cases where detailed context from specific segments of text is necessary,</p> <ul style="list-style-type: none"> • question answering • summarization.
Auto Merging Retrieval	<p>Extracting relevant passages from multiple sources and combining them into a cohesive context.</p> <p>Automatically combining passages into a single context.</p> <p>Feeding the merged context</p>	<p>Ideal for scenarios where information is scattered across different sources, like</p> <ul style="list-style-type: none"> • creating comprehensive reports • synthesis from diverse documents.
Recursive Retrieval	<p>Iteratively refining the retrieval process to improve relevance and accuracy.</p> <p>Using retrieved information to refine the input query or context.</p> <p>Iterative Retrieval: Repeating the retrieval process with refined queries or contexts.</p>	<p>Best suited for complex queries where initial retrievals may need further refinement, such as</p> <ul style="list-style-type: none"> • multi-turn dialogue systems • iterative problem-solving tasks • Pdfs with tabular data

Experiment Results

DATA : PDF documents, link mentioned on main page

Total queries : 50

Answer Similarity	Baseline Experiment	Sentence Window	Auto Merging	Recursive
Up train	25%	52%	54%	35%
Tonic Validate	44%	67%	68%	39%

Context Relevance	Baseline Experiment	Sentence Window	Auto Merging	Recursive
Up train	45%	69%	77%	33%
Tonic Validate	40%	74%	73%	55%

DATA : Code files from a GitHub repository, link mentioned on main page

Total queries : 10

Context similarity	Sentence window	Auto Merging	Recursive
Ragas	92%	96%	-
Tonic	4/5	4/5	-
Validate			
Deep Eval	46%	40%	-
Up Train	71%	65%	-
Tru lens	76.25%	76%	-
Falcon-evaluate	96	100%	-

Conclusion

- Every technique is better than the baseline experiment, and auto merging and sentence window both gave us quite good results!
- **Sentence Window Retrieval** : Good for extracting specific lines or sections from PR descriptions, comments, or diffs that are relevant to a query
- **Auto Merging Retrieval** : Useful for combining information from multiple PRs, comments, and reviews to provide a comprehensive response
- **Recursive Retrieval** : Highly suitable for iterative queries where initial results may need further refinement, such as detailed PR review or code analysis
- They are provided by framework llama index, and can be integrated with any LLM and embedding model
- For sentence window retrieval, what if we consider base nodes as well? This May increase the performance even more
- We can not test Recursive as the pipeline does not give us the retrieved context, and evaluation matrix required that to generate a score, but in pdf based testing, there are scores defined for it because the pipeline used partially satisfies its concepts, that's why it is highlighted red as well but still its better than baseline experiment.

Advance RAG : Context Rerankers

Overview

- The initial list of retrieved documents is reordered based on the new relevance scores generated by the re ranking model.
- The documents that are deemed more relevant in the given context are moved higher in the ranking.
- The top n ranked items for our experiments is kept four!
- The n retrieved items are kept six

Experiment Results

DATA : PDF documents, link mentioned on main page

Total queries : 50

Answer Similarity	Baseline Experiment	Colbert	Cohere	JIna	BGE	Rag Fusion	Rank Gpt	Long context reorder
Up train	25%	47%	49%	48%	33%	42%	49%	39%
Tonic Validate	44%	63%	54%	61%	51%	56%	60%	49%

Answer Similarity	Baseline Experiment	Colbert	Cohere	JIna	BGE	Rag Fusion	Rank Gpt	Long context reorder
Uptrain	45%	76%	67%	82%	43%	82%	78%	63%
Tonic Validate	40%	83%	77%	88%	66%	75%	87%	67%

DATA : Code files from a GitHub repository, link mentioned on main page

Total queries : 10

Context Relevance	Colbert reranker	Cohere reranker	Rag-fusion	RankGPT	Bge ranker	Long context reorder	JIna Ai reranker
Ragas	82%	90%	87%	88%	82.5%	93.0%	93%
Tonic Validate	4.5/5	3.2/5	3.6/5	3.8/5	3.4/5	4/5	3.8/5
Deepeval	70%	40%	68%	50%	33%	45.8%	63%
Uptrain	70%	39%	51%	53%	33%	45.2%	48%
Trulens	75.68%	75.68%	75.6%	76.25%	76.22%	76.22%	76.01%
Falcon-evaluate	96%	56%	42%	57%	55%	42%	42%

Conclusion

- All rerankers performed better than baseline experiment.
- Colbert re ranker provides a comparison between a retrieval score and a ranker score, for all cases ranker score is lower than retrieval score
- Cohere re-rank contradicts its score with relevancy score, as score difference is huge!
- Rag Fusion allows you to re-rank context from different rewritten queries as well but it decreases the scores when you have rewritten queries, mainly cause of the quality of rewritten query's, can change it to sub queries method too, as the quality from it is concluded to be good
- Rankgpt provides you with new ranked order of contexts like 1,2,3,4,..n
- Long context reorder does not provide scores, also gives window issues when pass to llm for QA, typically because of too long contexts, so k is limited to 3 in this case
- BGE reranker pipeline is customized and ran on t4-GPU, also it gives non normalised scores
- Bert base rerankers have a better answer relevance.
- Colbert re ranker seems to perform better than other rerankers, also provides a differences for estimation.
- Colbert, jina re ranker both standout in their scores, and both of them don't rely on llm for re ranking cross encoders based re ranker are yielding good results than the ones which are based on llm - optimized for a pair (query to document) - you get a better score, and if your evaluator is checking only top few relevant documents, it will rate cross encoders based re ranker a high score

Comparing Rerankers response time on GPU and CPU

Re-ranker	CPU Avg Score	GPU Avg Score
Colbert	44s	60s
Cohere	61s	120s
Rag Fusion	80s	60s

Rank GPT	300s	180s
Jina AI	47s	60s
BGE	-	9s
Long Context Reorder	58s	180s

- Bert based Rerankers took less time on CPU, Bert architecture does involve matrix multiplication as well as layered operations and % GPT based Rerankers took less time on GPU
- **BERT-based models:** Their smaller size, bidirectional nature, and extensive CPU optimizations make them efficient on CPU's.
- **GPT-based models:** Their larger size, auto regressive nature, and the need for extensive parallel processing make them better suited for GPU s.

Testing Frameworks

TruLens: TruLens offers complex pipelines that vary for each metric, making integration into RAG pipelines challenging. While you could build a RAG pipeline already integrated with TruLens, this approach limits the ability to incorporate multiple advanced RAG tools. The accuracy score provided by TruLens is inconsistent, typically fluctuating between 74-76%, which undermines reliability.

Falcon Evaluate: Falcon Evaluate provides an extensive array of metrics and is unique in utilizing traditional ML models for evaluation. However, its performance on negative testing is subpar, producing inaccurate results. Additionally, its accuracy scores are erratic, often falling between 40-60%, and if a response is exceptionally good, the score jumps above 90%. It rarely outputs scores between these extremes, indicating a lack of nuance in its evaluations.

Ragas: Usually becomes inconsistent, higher values of responses are observed when other frameworks are giving low score for similar response

DeepEval, Tonic Validate, Uptrain: These frameworks offer essential metrics for evaluation and are easily integrable into custom pipelines. If ranked by performance and consistency, the order would be:

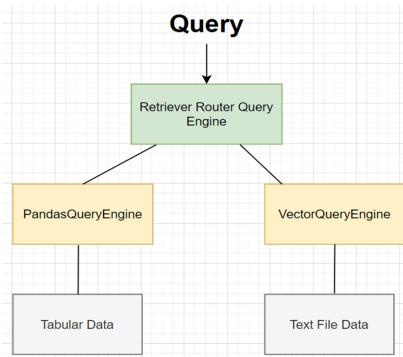
1. Tonic Validate
2. Uptrain
3. Deepeval

The results from these frameworks are not directly comparable due to the use of different models, threshold values, and formulas for calculating metrics.

RAG over different types of Data

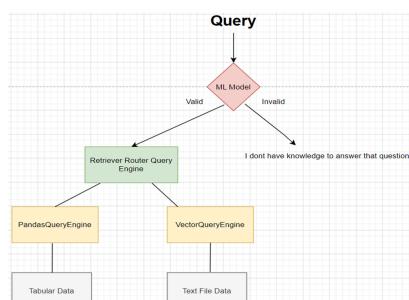
Key Findings

- Implementing Retrieval Augmented Generation over different types of data, e.g if underlying data contains pdf documents, SQL database, JSON files etc. Exploring how can we create a single RAG over that.
- Llama index provides the concept of multiple query engines, each specifically designed to work with a certain type of data. For example, if you have SQL, JSON, tabular data, then there are different query engines specifically made to handle each kind of data.
- Llama index also provides a Retriever Router Query engine which can be used to route an incoming question to the right query engine.
- The idea was to develop different query engines specific to each data, then plate the RetrieverRouterEngine on top of them and observe whether it routes the question to the correct query engine or not.
- In my example, I built a PandasQueryEngine over a tabular data, a VectorQueryEngine over a pdf data, placed the RetrieverRouterQuery engine on top of them and experimented whether it routes the question to the proper QueryEngine or not.



Results

- For the most part, the RetrieverRouterQuery engine seemed to be directing the query to the correct QueryEngine.
- An issue that was originating was that the LLM was not restricting its responses to the knowledge base that we had provided. Instead it was generating responses from its own knowledge. To resolve this, we need to adhere to "Strict Prompting"
- Another issue was how to restrict unrelated questions from going down the pipeline, because RetrieverRouterQuery engine tries to fit an incoming question into one of the query engines, so even if a question that is not related to the knowledge about which we have created our RAG system comes in, it will eventually be passed into a query engine which will try to answer it.
- To cater this problem, we can either use an LLM or an ML model to filter out incoming questions, and pass only the relevant one down the pipeline.



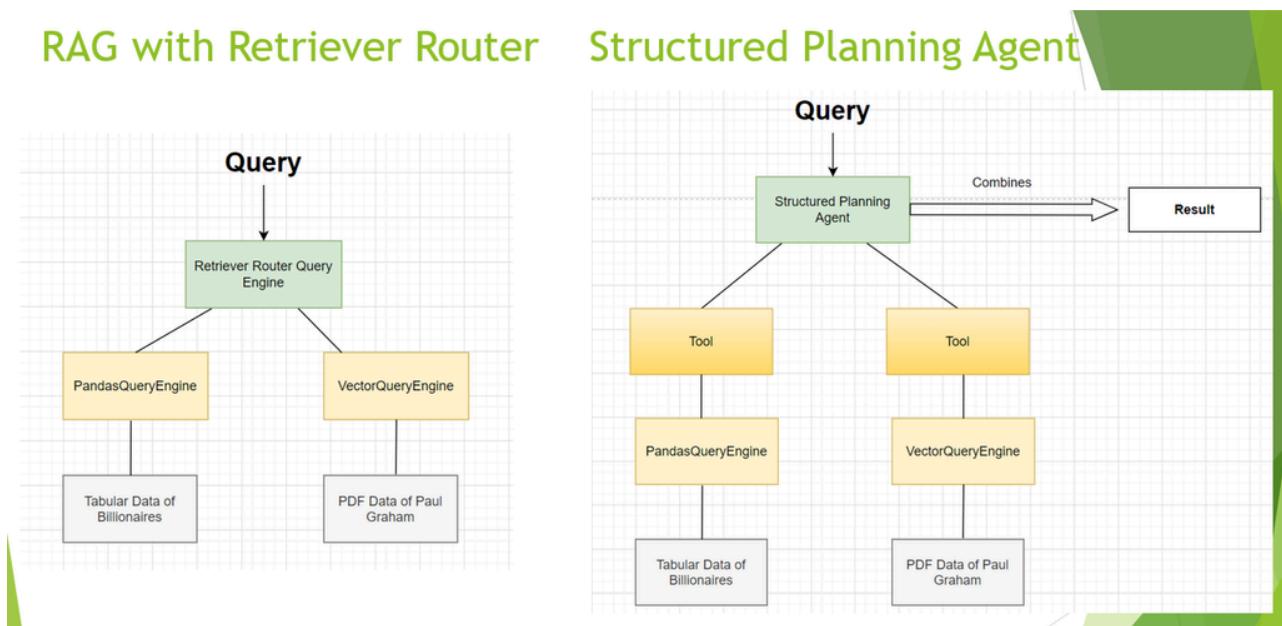
References

- [!\[\]\(57e6eaa370c11c0a8b197170cf41ea3b_img.jpg\) Pandas Query Engine - LlamaIndex](#)
- [!\[\]\(dbb5925b453b6e9e7a1719382db80907_img.jpg\) Retriever Router Query Engine - LlamaIndex](#)

Structured Planning Agent Llama-index

Structured Planning Agent

- Structured Planning Agent essentially works by resolving the incoming query into a list of sub tasks, executing each sub task through an appropriate tool and then combining the results.



- The above image illustrates the difference between RAG with RetrieverRouterQuery engine and Structured Planning Agent. The Retriever Router analyzes the incoming query and forwards it to **ONE** of the underlying query engines, whereas a Structured Planning Agent break down the incoming query into sub tasks and uses appropriate tools to call relevant underlying query engines to process the sub queries, then combines the results.
- Retriever Router Query Engine is suitable for queries like:
 - “What was the net worth of Bill Gates?”
 - This information is present in the tabular data of billionaires (see image above) and only needs access to that.
 - “Describe the career of Paul Graham.”
 - This information is present in the PDF doc of Paul Graham and only needs access to that.
- On the other hand, Structured Planning Agent is suitable for following queries:
 - “Tell me the net worth of Bill Gates and describe the career of Paul Graham.”
 - One single data source does not contain information to answer the above question. Structured Planning Agent will break down the query into sub tasks:
 - 1) Get net worth of Bill Gates.
 - 2) Get career information regarding Paul Graham.
 - Then use tool1 built on tabular data of billionaires to answer query1 and use tool2 built on PDF data of Paul Graham to answer query2.
 - How does the system know which tool and query engine to use to answer which sub query? We specify that information while building the system. In the image below, while building the tool we associate the relevant query engine with it along with a description about which kind of data it is useful to answer a question from.

```

uber_index = VectorStoreIndex.from_documents(uber_documents)

lyft_tool = QueryEngineTool.from_defaults(
    df_query_engines[0],
    name="billionaires",
    description="Useful for asking questions about billionaires and their net worth.",
)

uber_tool = QueryEngineTool.from_defaults(
    uber_index.as_query_engine(text_qa_template=text_qa_template, refine_template=refine_template),
    name="paul_graham",
    description="Useful for asking questions about paul graham.",
)

```

- After retrieving the results, the Structured Planning Agent combines them and outputs a single answer.

Key Takeaways

- Structured Planning Agent works well only with a capable LLM. This is because under the hood, a lot of reasoning is happening in order to select the appropriate tools so it is suggested to use the best available llm.
- During experimentation, llama3-70b was not performing well. When GPT4-turbo, which is a better model, was used the results were significantly better.
- Because of agentic reasoning, this approach is slightly more time consuming in yielding answers.

References

-  [Structured Planning Agent - Llamaindex](#)

Explore Evaluation Metrics

Giskard:

Automatic evaluation methodology. Covers automatic evaluation as well as manual evaluation by providing human reviewers to evaluate the results provided through automatic evaluation.

Can be used in identifying hallucinations or any kind of harmful responses or legal issues (in case the model generates)

Why G-eval and RAGAS produced different results?

G-Eval:

- Coherence: G-Eval is designed to evaluate specific criteria such as coherence through detailed prompts and chain-of-thoughts (CoT). It focuses on how well the generated text maintains a logical and structured flow.
- Prompt-Based Evaluation: The evaluations are driven by a predefined prompt that specifies what aspects of the text to evaluate, such as coherence, conciseness, or grammar. This can make G-Eval sensitive to the specific wording and focus of the prompts.
- Scoring Function: The scoring function in G-Eval uses the probabilities of output tokens from LLMs, aiming to provide fine-grained, continuous scores.

RAGAS:

- Faithfulness: Measures how well the answer is grounded in the given context, which is crucial for ensuring that the information is factually correct and relevant to the retrieved context.
- Answer Relevance: Evaluates if the generated answer addresses the given question appropriately without considering factuality.
- Context Relevance: Assesses the relevance of the retrieved context, focusing on minimizing redundant information and maximizing the usefulness of the context provided to answer the question.

G-Eval: Primarily designed for Natural Language Generation (NLG) tasks, it may weigh criteria like coherence and conciseness more heavily. The auto CoT provides a step-by-step approach that focuses on logical flow and structure.

RAGAS: Designed for Retrieval-Augmented Generation (RAG) systems, it emphasizes the factual consistency of the generated text with the retrieved context (faithfulness), how well the generated answer addresses the question (answer relevance), and the pertinence of the retrieved context (context relevance).

G-Eval: Since it is prompt-based, the quality and detail of the prompt can significantly affect the evaluation outcome.

RAGAS: Utilizes predefined prompts for extracting statements and assessing relevance and faithfulness.

G-Eval: Evaluates context expansion by how well it improves the coherence, structure, and overall quality of the generated text. If context expansion leads to more logically structured and coherent text, G-Eval might score it higher.

RAGAS: Focuses on how context expansion impacts the faithfulness, relevance of the answer, and relevance of the context. If the expanded context improves the factual consistency and relevance without adding redundancy, RAGAS would score it higher.

1. Coherence vs. Faithfulness: G-Eval might score context expansion highly if it improves the logical flow and coherence of the text, even if the expanded context introduces some factual inconsistencies. RAGAS, on the other hand, might penalize the same context expansion if it perceives a decrease in faithfulness or an increase in irrelevant information.
1. Prompt-Based Nuances: The way prompts are constructed in G-Eval can lead to different evaluation outcomes based on the specific criteria emphasized in the prompt. RAGAS's structured approach to extracting and verifying statements might provide a different perspective on the same content.
1. Scoring Granularity: G-Eval's use of token probabilities for scoring might result in more nuanced scores compared to RAGAS's binary verification process, which might lead to discrepancies in the final evaluation.

TruLens

Use Cases:

- Detect and Mitigate Hallucination
- Improve Retrieval Quality
- Optimize App Configuration
- Verify the Summarization Quality

Task	Evaluation parameters
Content moderation	Recall and precision on toxicity and bias
Query generation	Correct output syntax and attributes, extracts the right information upon execution
Dialogue (chatbots, summarization, Q&A)	Faithfulness, relevance

Arize AI Phenix:  [Quickstart: Evals | Phoenix](#)

The standard for evaluating text is human labeling. However, high-quality LLM outputs are becoming cheaper and faster to produce, and human evaluation cannot scale. In this context, evaluating the performance of LLM applications is best tackled by using a separate evaluation LLM. The Phoenix LLM Eval Library is designed for simple, fast, and accurate LLM-based evaluations.

DeepCheck:  [LLM Evaluation Solutions | Deepchecks](#)

Helps in reducing the risk in llm pipelines by flagging significant issues and also helps in safeguarding the model from the problem of hallucinations.

It also provides auto-annotation which helps in understanding whether the response is good or bad.

Evaluation Metrics

Giskard:

Automatic evaluation methodology. Covers automatic evaluation as well as manual evaluation by providing human reviewers to evaluate the results provided through automatic evaluation.

Can be used in identifying hallucinations or any kind of harmful responses or legal issues (in case the model generates)

Why G-eval and RAGAS produced different results?

G-Eval:

- Coherence: G-Eval is designed to evaluate specific criteria such as coherence through detailed prompts and chain-of-thoughts (CoT). It focuses on how well the generated text maintains a logical and structured flow.
- Prompt-Based Evaluation: The evaluations are driven by a predefined prompt that specifies what aspects of the text to evaluate, such as coherence, conciseness, or grammar. This can make G-Eval sensitive to the specific wording and focus of the prompts.
- Scoring Function: The scoring function in G-Eval uses the probabilities of output tokens from LLMs, aiming to provide fine-grained, continuous scores.

RAGAS:

- Faithfulness: Measures how well the answer is grounded in the given context, which is crucial for ensuring that the information is factually correct and relevant to the retrieved context.
- Answer Relevance: Evaluates if the generated answer addresses the given question appropriately without considering factuality.
- Context Relevance: Assesses the relevance of the retrieved context, focusing on minimizing redundant information and maximizing the usefulness of the context provided to answer the question.

G-Eval: Primarily designed for Natural Language Generation (NLG) tasks, it may weigh criteria like coherence and conciseness more heavily. The auto CoT provides a step-by-step approach that focuses on logical flow and structure.

RAGAS: Designed for Retrieval-Augmented Generation (RAG) systems, it emphasizes the factual consistency of the generated text with the retrieved context (faithfulness), how well the generated answer addresses the question (answer relevance), and the pertinence of the retrieved context (context relevance).

G-Eval: Since it is prompt-based, the quality and detail of the prompt can significantly affect the evaluation outcome.

RAGAS: Utilizes predefined prompts for extracting statements and assessing relevance and faithfulness.

G-Eval: Evaluates context expansion by how well it improves the coherence, structure, and overall quality of the generated text. If context expansion leads to more logically structured and coherent text, G-Eval might score it higher.

RAGAS: Focuses on how context expansion impacts the faithfulness, relevance of the answer, and relevance of the context. If the expanded context improves the factual consistency and relevance without adding redundancy, RAGAS would score it higher.

1. Coherence vs. Faithfulness: G-Eval might score context expansion highly if it improves the logical flow and coherence of the text, even if the expanded context introduces some factual inconsistencies. RAGAS, on the other hand, might penalize the same context expansion if it perceives a decrease in faithfulness or an increase in irrelevant information.
1. Prompt-Based Nuances: The way prompts are constructed in G-Eval can lead to different evaluation outcomes based on the specific criteria emphasized in the prompt. RAGAS's structured approach to extracting and verifying statements might provide a different perspective on the same content.
1. Scoring Granularity: G-Eval's use of token probabilities for scoring might result in more nuanced scores compared to RAGAS's binary verification process, which might lead to discrepancies in the final evaluation.

TruLens

Use Cases:

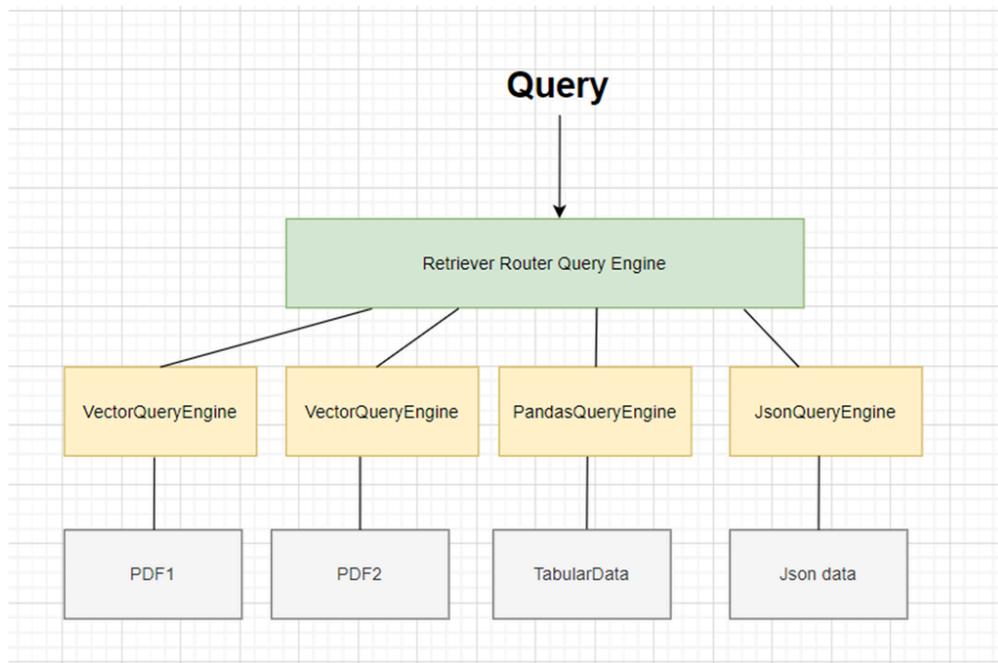
- Detect and Mitigate Hallucination
- Improve Retrieval Quality
- Optimize App Configuration
- Verify the Summarization Quality

Task	Evaluation parameters
Content moderation	Recall and precision on toxicity and bias
Query generation	Correct output syntax and attributes, extracts the right information upon execution
Dialogue (chatbots, summarization, Q&A)	Faithfulness, relevance

Q/A with RetrieverRouter engine from llama-index

Key Points

- Using the llama-index retriever router query engine to observe its efficacy in routing the incoming question to the correct query engine below.



- The image above shows the experimental setup.
- PDF1 contains information regarding Emumba Provident fund policy. PDF2 is related to Titanic. Tabular data consists of a table which has names, net worths and nationalities of top billionaires. Json data contains dummy data related to employees.
- Each kind of data has a respective query engine from llama-index built on top of it.
- Finally, the Retriever Router Query engine is used to route the incoming question to the correct query engine.

Results

- The results achieved were not very promising.
 - The system was routing the question to the incorrect query engine at times.
 - It was forcing to answer the question using one of the underlying query engines despite the question being out of context.
 - It does not have the ability to break down the question and use multiple query engines to answer parts of it, if that is required.
- Primarily, it is most evident that Retriever Router Query engine is designed for systems built on the same underlying data but different types of indexes, rather than systems which are built on different types of data.

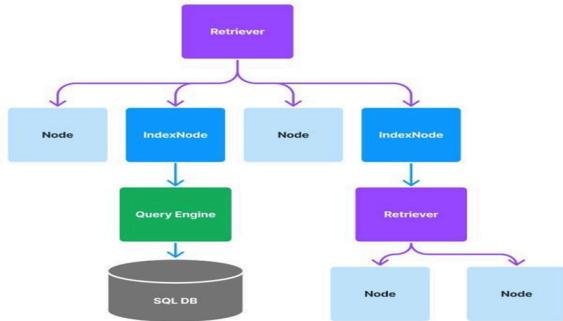
References

- [!\[\]\(1abe51175dd8ebafd0041731b2dc5fe9_img.jpg\) Retriever Router Query Engine - LlamaIndex](#)

Recursive Retrieval Techniques

Key Takeaways

- Recursive retrieval is beneficial when dealing with hierarchical data or unstructured forms of data, for example data in a tabular form,
- Has the concept of IndexNode which contains NodeData+underlying query engine.
- During retrieval, if an IndexNode is retrieved, the underlying query engine is called which gets executed.



– QueryEngines:

The concept of query engines is interesting. In general, we've observed that LLMS are not very effective at answering questions from structured forms of data (e.g a table) effectively. Consider an example table containing information regarding billionaires. We want to answer the question "How many people are from India and France" using the data of the following table. (Image1 → Table , Image2 → Query and Response)

table_df[0]					
No.	Name	Net worth\n(USD)	Age	Nationality	Primary source(s) of wealth
0	Bernard Arnault &\nfamily	\$233 billion	75	France	LVMH
1	Elon Musk	\$195 billion	52	South Africa\nCanada\nUnited\nStates	Tesla, SpaceX, Twitter (Currently\nX)
2	Jeff Bezos	\$194 billion	60	United\nStates	Amazon
3	Mark Zuckerberg	\$177 billion	39	United\nStates	Meta Platforms
4	Larry Ellison	\$141 billion	79	United\nStates	Oracle Corporation
5	Warren Buffett	\$133 billion	93	United\nStates	Berkshire Hathaway
6	Bill Gates	\$128 billion	68	United\nStates	Microsoft
7	Steve Ballmer	\$121 billion	68	United\nStates	Microsoft
8	Mukesh Ambani	\$116 billion	65	India	Reliance Industries
9	Larry Page	\$114 billion	51	United\nStates	Google

```
response = df_query_engines[0].query(  
    "How many people are from France and India ?"  
)  
print(str(response))  
response  
2  
Response(response='2', source_nodes=[], metadata={'pandas_instruction_str': '(df["Nationality"].str.contains("France|India")).sum()', 'raw_pandas_output': '2'})
```

Now inputting the content of the table and asking the LLM this question directly does not yield good results. LLMS are not great at manoeuvring around this tabular data.

Solution ? : Llamaindex query engines !!

In this specific case we can use pandasQueryEngine from Llamaindex. Under the hood, it inputs the dataframe column names and asks it to write python code to extract the information (rather than giving a direct answer). This yields significantly better result than asking the LLM to answer the question directly.

- Similarly, there are specific query engines available for each specialised task (JSONQueryEngine for JSON related data, SQLEngine for SQL).

References

→  Recursive Retriever + Query Engine Demo - LlamaIndex

→  Advanced RAG: Recursive Retrieval with llamaindex

→  Jerry Liu on Twitter / X

Jerry Liu on Twitter / X

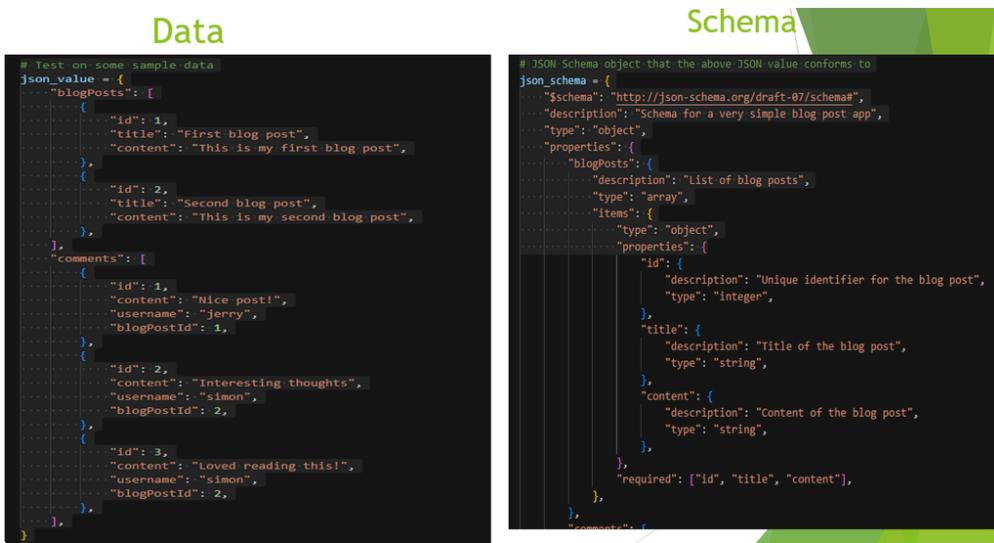
Exploring JSON query engines of llama-index

Key Points

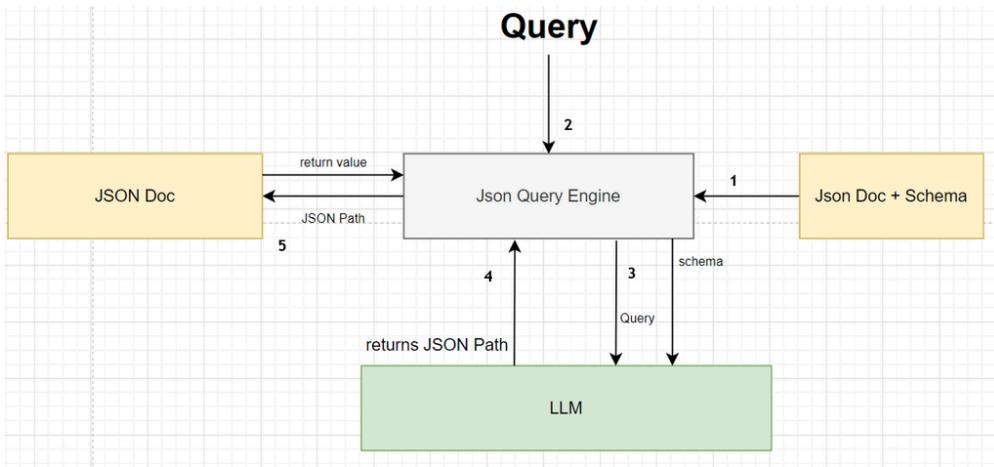
- Llama-index provides two kinds of query engines to query JSON data.
(JSON Query Engine , JSONAnalyze)

JSON Query Engine

- It is ideally used over a nested JSON data object with a JSON schema as shown in the picture below.



- In the image below, we can understand how does the JSON query engine works under the hood.



- 1) We create the query engine by providing it with the schema and the data
- 2) At runtime, a query is sent to the query engine to be processed. (e.g What did Jerry comment ?)
- 3) The JSON Query engine passes the query and the JSON Schema to the LLM and asks it to generate a valid JSON Path.
- 4) LLM returns a valid JSON Path according to the question provided.
(e.g `$.comments[?(a.username=='jerry')]`)

- 5) JSON Query engine executes that JSON Path on the JSON doc to retrieve the answer of the question.

(e.g the answer to the above question : "Nice Post!")

Results:

- For simple, straightforward questions that involved answering only a single piece of information, the JSON query engine worked fine.
(e.g Question: Who commented "Loved reading this!"?)

```

query = "Who commented Loved reading this!?"

nl_response = nl_query_engine.query(
    query,
)
raw_response = raw_query_engine.query(
    query,
)
[4] ✓ 7.4s

print("Natural Language Response: ", nl_response)
print("Raw Response: ", raw_response)
[5] ✓ 0.0s
...
Natural Language Response: The user 'simon' commented 'Loved reading this!'.
Raw Response: {"username": "simon"}
```

- For slightly more complex questions, where multiple pieces of information need to be fetched from the JSON doc and an 'and' or an 'or' condition is required under the hood, the JSON Query engine doesn't respond well. See image below. "What did jerry and simon comment?" was asked.

```

query = "What did jerry and simon comment?"

nl_response = nl_query_engine.query(
    query,
)
raw_response = raw_query_engine.query(
    query,
)
[6] ⏺ 3.6s

-----
JsonPathParserError
Traceback (most recent call last)
File c:\Users\Emumba\AppData\Local\Programs\Python\Python312\Lib\site-packages\llama_index\core\indices\s
  82 try:
--> 83     datum: List[DatumInContext] = parse(expression).find(json_value)
  84     if datum:
File c:\Users\Emumba\AppData\Local\Programs\Python\Python312\Lib\site-packages\jsonpath_ng\ext\parser.py:
  123 def parse(path, debug=False):
--> 124     return ExtendedJsonPathParser(debug=debug).parse(path)

File c:\Users\Emumba\AppData\Local\Programs\Python\Python312\Lib\site-packages\jsonpath_ng\parser.py:38,
  37 lexer = lexer or self.lexer_class()
--> 38 return self.parse_token_stream(lexer.tokenize(string))

File c:\Users\Emumba\AppData\Local\Programs\Python\Python312\Lib\site-packages\jsonpath_ng\parser.py:62,
  54 new_parser = ply.yacc.yacc(module=self,
  55                             debug=self.debug,
  56                             tabmodule = parsing_table_module,
(...),
  59                             start = start_symbol,
  60                             errorlog = logger)
--> 62 return new_parser.parse(lexer = IteratorToTokenStream(token_iterator))

File c:\Users\Emumba\AppData\Local\Programs\Python\Python312\Lib\site-packages\ply\yacc.py:333, in LRParser
...
  89     except Exception as exc:
--> 90         raise ValueError(f"Invalid JSON Path: {expression}") from exc
  92 return results

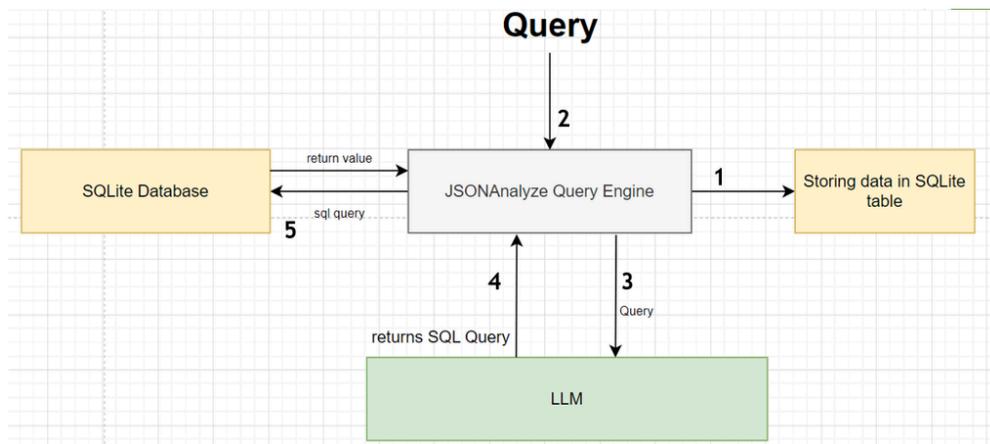
ValueError: Invalid JSON Path: $.comments[?(@.username=='jerry' || @.username=='simon')].content
Output truncated. View the full output by running this command in a terminal window: python -m llama_index --verbose
```

JSONAnalyze Query Engine

- JSONAnalyze Query engine is provided by llama-index to query over typical JSON data that is retrieved from an api(see example in image below). JSONAnalyze is helpful is answering statistical questions over this JSON data.

```
json_list = [
    {
        "name": "John Doe",
        "age": 25,
        "major": "Computer Science",
        "email": "john.doe@example.com",
        "address": "123 Main St",
        "city": "New York",
        "state": "NY",
        "country": "USA",
        "phone": "+1 123-456-7890",
        "occupation": "Software Engineer",
    },
    {
        "name": "Jane Smith",
        "age": 30,
        "major": "Business Administration",
        "email": "jane.smith@example.com",
        "address": "456 Elm St",
        "city": "San Francisco",
        "state": "CA",
        "country": "USA",
        "phone": "+1 234-567-8901",
        "occupation": "Marketing Manager",
    },
    {
        "name": "Michael Johnson",
        "age": 35,
        "major": "Finance",
        "email": "michael.johnson@example.com",
        "address": "789 Oak Ave",
    }
]
```

- In the image below, we can see how JSONAnalyze query engine works under the hood.



- 1) We create the query engine by providing it with JSON Data and it stores that data in an SQLite database.
- 2) At runtime, a query is sent to the query engine to be processed. (e.g What is the distribution of ages among the individuals ?)
- 3) The JSONAnalyze Query engine passes the query and the SQL database schema to the LLM and asks it to generate a valid SQL Query.
- 4) LLM returns a SQL Query according to the question provided.
- 5) JSONAnalyze Query engine executes that SQL query on the SQLite table to retrieve the answer of the question.

Results:

- For most of the questions that conform to the fields and available data in the JSON object (which is converted to an SQLite table under the hood), JSONAnalyze query engine performs well. In the image below, it was able to answer all the these questions without giving an error.

```

questions = [
    "What is the average age of the individuals in the dataset?",
    "What is the maximum age among the individuals?",
    "What is the minimum age among the individuals?",
    "How many individuals have a major in Psychology?",
    "What is the most common major among the individuals?",
    "What is the percentage of individuals residing in California (CA)?",
    "How many individuals have an occupation related to science or engineering?",
    "What is the average length of the email addresses in the dataset?",
    "How many individuals have a phone number starting with '+1 234'?",
    "What is the distribution of ages among the individuals?",
]

# print(json_stats_query_engine.query(questions[-1]))
print(json_stats_query_engine.query(""))

# print(query_engine.query(questions[0]))
✓ 3.5s

The distribution of ages among the individuals is as follows:
- Age 25: 1 individual
- Age 26: 1 individual
- Age 27: 2 individuals
- Age 28: 3 individuals
- Age 29: 2 individuals
- Age 30: 2 individuals
- Age 31: 1 individual
- Age 32: 2 individuals
- Age 33: 1 individual
- Age 35: 1 individual

```

- But for questions that contain any kind of information that is not present in the JSON, JSONAnalyze query engine will through an error.

```

> ^
# print(json_stats_query_engine.query(questions[-1]))
print(json_stats_query_engine.query("How many people are from company 'abcdef'?"))

# print(query_engine.query(questions[0]))

11] 3.1s

.. Error executing query: SELECT COUNT(*)
FROM items
WHERE company = 'abcdef';

-----
OperationalError                                Traceback (most recent call last)
File c:\Users\Emumba\AppData\Local\Programs\Python\Python312\Lib\site-packages\llama_index\co
    105     try:
    106         # Execute the SQL query
--> 107         results = list(db.query(sql_query))
    108     except sqlite_utils.utils.sqlite3.OperationalError as exc:
File c:\Users\Emumba\AppData\Local\Programs\Python\Python312\Lib\site-packages\sqlite_utils\d
    496     """
    497     Execute ``sql`` and return an iterable of dictionaries representing each row.
    498
    (...)

    501     parameters, or a dictionary for ``where id = :id``
    502     """
--> 503     cursor = self.execute(sql, params or tuple())
    504     keys = [d[0] for d in cursor.description]

File c:\Users\Emumba\AppData\Local\Programs\Python\Python312\Lib\site-packages\sqlite_utils\d
    520     if parameters is not None:
--> 521         return self.conn.execute(sql, parameters)
    522     else:

OperationalError: no such column: company
...
    109     print_text(f"Error executing query: {sql_query}")
--> 110     raise ValueError("Invalid query") from exc
    112 return sql_query, table_schema, results

ValueError: Invalid query
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```

In the image above, the question mentions company name, so the LLM generates an SQL query containing a filter on company name, but the original data doesn't contain any field corresponding to a company name, hence when the JSONAnalyze query engine executes this SQL query on the SQLite database, it throws an error.

References

- [💡 JSON Query Engine - LlamalIndex](#)
- [💡 JSONAnalyze Query Engine - LlamalIndex](#)

Capture Explanations and Compare Our RAG Mindmap

Reference: [10 Challenges in Building RAG-Based LLM Applications - YouTube](#)

Customized Models:

- Techniques: Fine-tuning, modifying, transfer learning, etc.

Alternate Approach:

- Turn inference problem into retrieval plus inference problem aka semantic search.
- Utilize traditional information retrieval to provide context.

Stage One (Data Ingestion):

1. Data Ingestion:

- Store data efficiently for retrieval.
- Break down into chunks for manageable processing.
- Convert documents into vector representations using an encoder.
- Store embedding vectors in a vector database.

2. Challenges:

- Data quality issues (garbage data, sensitive data, etc.).
- Inaccurate data ingestion (e.g., tables, diagrams, different formats).
- Loss of continuity and high-level context due to document chunking.
- Dependency on embedding model quality.
- Unambiguous or incomplete user queries.
- Incorrect document retrieval and utilization (hallucination).

Data Ingestion:

1. Data Preprocessing:

- Data loading and quality checks.

2. Chunking:

- Strategies: Document-based, semantic, recursive.
- Selection depends on knowledge base nature, model capabilities, and task.

3. Embeddings.

4. Scalable Data Ingestion:

- Consider scalability and planning.
- Data cleaning for large datasets.

5. Indexing in Vector Database.

Pre-Retrieval:

- Querying Challenges:
 - Handling ambiguous queries.
- Solution:
 - Query optimisation techniques like
 - Query transformations
 - Sub query decomposition
 - Step back prompting
 - Multi query retrieval

- Multi step retrieval
- Query rewriting
- Query Routing.

Retrieval:

- Challenges:

- Too many or too few chunks.
- Missing context, irrelevant chunks, query ambiguity.
- Semantic ambiguity, vector similarity issues.
- Retrieval optimizations.

- Solutions:

- Hybrid search
- Metadata integration
- Sentence window retrieval
- Hypothetical document embedding
- Parent child chunk retrieval
- Hierarchical index retrieval
- Multi document agents
- Chat engine context
- Chat engine condense plus context

Problem: Retrieval of Irrelevant Chunks.

- Post-Retrieval:

- Reranking to address information overload and hallucination.

- Solutions:

- Utilize re-rankers like cross encoder reranking, cohore, private reranking APIs.

Generation:

- Problems:

- Information overload, insufficient context window.
- Domain adaptation, chaotic contexts.
- Structuring retrieved information, hallucination, irrelevant responses.
- Logically incoherent responses.

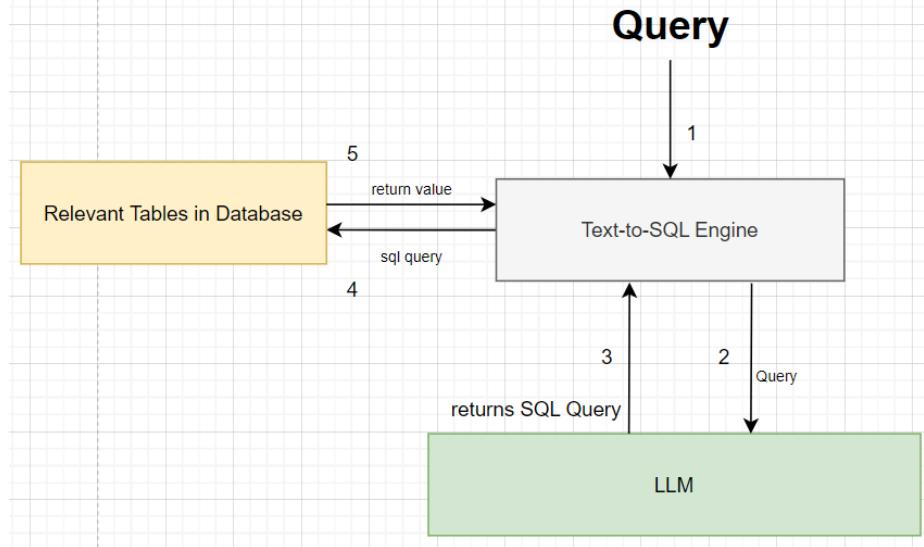
- Solutions:

- Information compression
- Generator fine-tuning
- Adapter methods
- Throat of thought
- Chain of note
- Chain of verification
- React (reason + act)
- Expert prompting.

Exploring text-to-SQL Query engine of llama-index

Key Points

- Llama-index text-to-SQL Query engine is designed to answer questions over SQL data effectively.
- It follows the following pipeline (which other techniques of q/a over structured data follow as well)



1. The user's question, and the schema of the table are sent to the LLM.
2. LLM is prompted to generates an SQL query to answer the user's question.
3. LLM generates an SQL query to be run.
4. The generated SQL query is executed on the relevant table.
5. The returned value is formatted into a natural language response.

References

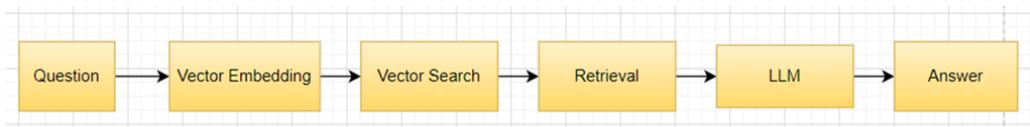
- [Text-to-SQL Guide \(Query Engine + Retriever\) - LlamaIndex](#)

Q/A over unstructured vs structured data

Key Points

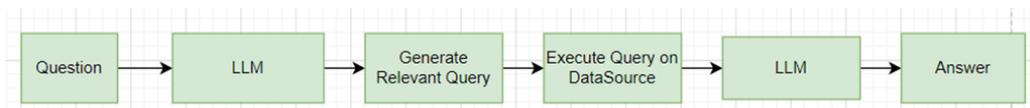
- Lately, two different techniques and pipelines are being used in order to perform Q/A over structured and unstructured data. They utilise the power of LLMs in different ways.

Unstructured Data



- Chat with unstructured data (e.g PDFs, word files etc) is done through standard Retrieval Augmented Generation. The pipeline is shown in the above image.
- The user's question is converted into a vector embedding (using the same embedding model through which the document chunks were converted into embeddings).
- There's a similarity search (e.g cosine similarity) between the question's vector and vectors of all chunks in the vector database.
- The most similar chunks are retrieved from the vector database
- The embeddings of the question and the chunks are converted back into their textual forms and then fed into an LLM to answer the question.

Structured Data



- Chat with structured data (Tabular data, SQL data, JSON) does not involve the traditional RAG pipeline because, fundamentally, its underlying workings do not favour Q/A over structured data.
- The idea is that rather than asking the LLM to directly answer a question from structured data, we ask the LLM to write a data-specific query and then execute that query separately on the data.

For example, if we have an SQL table with columns "Countries,Population" and we have a question "What is the total Population of the top three most populous countries ?"

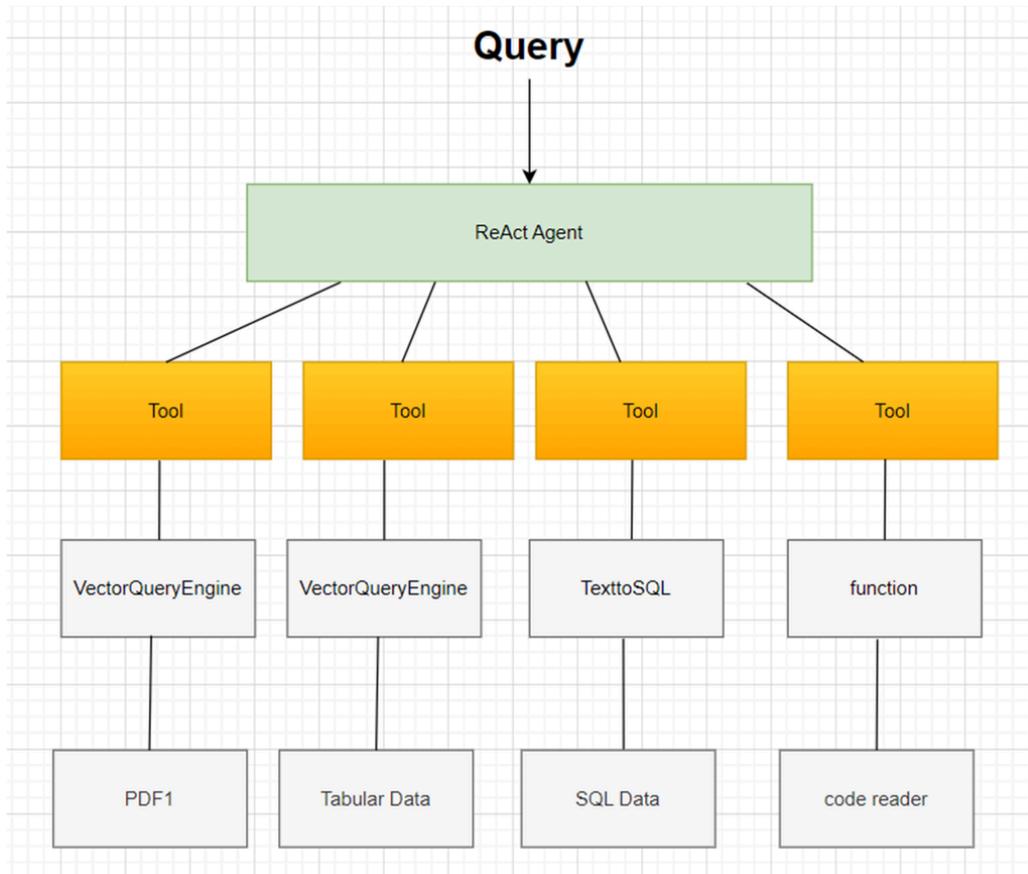
Now rather than asking the LLM this question directly, we ask the LLM to generate an SQL query to answer the question, and we execute that query separately in the table to get the answer.

- The image above illustrates this flow.
- There is
 - no conversion of data into chunks
 - no storage of chunks' embeddings in vector databases
 - no conversion of runtime question into vector embedding
 - no similarity index matching and top-k retrieval from database
- All of the above mentioned steps are part of standard RAG which performs well on unstructured data.

Q/A with ReAct agent for routing

Key Points

- Using the ReAct agent as a query routing entity. Its main purpose is to route the incoming question from the user to the correct underlying tool/function.
- Of all the query routing mechanisms tested, ReAct gives the most promising results.



- The image above shows the setup.
- A vector query engine built on PDF. This is a RAG pipeline. (PDF data is about Emumba's policies mentioned on internal site)
- A vector query engine built on Tabular data (Emumba's info regarding OPD centers).
- A text-to-sql query engine built on SQL data. (Dummy data containing a table with countries and their population information and another table containing data about employees).
- A code-reader function which reads code from a file. Is added to the agent to test whether Q/A works alongside agentic functionality.

Results

- The results were were promising.
- The following images illustrate correct responses from the above system.

(Verbose==True is set so that we can see how the ReAct agent is reasoning)

Prompt: What is the percentage of provident fund provided by emumba ?
Thought: The user is asking about the provident fund percentage provided by Emumba. I should use the perks_and_benefits tool to get this information.
Action: perks_and_benefits
Action Input: {'input': 'provident fund'}

Observation: The Provident Fund is a savings plan for all permanent employees, allowing them to save a portion of their salary each month. Both the employer and employee contribute 7.5% of the gross salary to this fund. Employees become eligible for this benefit upon the confirmation of their probation completion. If an employee resigns within the first year, they only receive their own contribution. After one year, they are eligible for 50% of the employer's contribution, after two years, 75% of the employer's contribution, and after three years, 100% of the employer's contribution. Employees who contribute to a recognized Provident Fund also receive tax exemption benefits. To enroll in the Provident Fund benefit program, employees need to fill out a PF Consent Form and submit it to the Finance Manager. However, due to a government policy, the Provident Fund is currently not being recognized.

Thought: I can answer without using any more tools. I'll use the user's language to answer.

Answer: Emumba provides a Provident Fund for all permanent employees, where both the employer and the employee contribute 7.5% of the gross salary each month. However, due to a government policy, the Provident Fund is currently not being recognized.

Emumba provides a Provident Fund for all permanent employees, where both the employer and the employee contribute 7.5% of the gross salary each month. However, due to a government policy, the Provident Fund is currently not being recognized.

Emumba provides a Provident Fund where both the employer and the employee contribute 7.5% of the gross salary.

Prompt: Does emumba provide lunch ?

Thought: The user is asking about the perks and benefits provided by Emumba. I should use the perks_and_benefits tool to answer this question.

Action: perks_and_benefits
Action Input: {'input': 'Does emumba provide Lunch ?'}

Observation: Yes, Emumba provides lunch. They offer a wide variety of tasty, high-quality meals at a subsidized price. The menu changes every three months and includes a variety of Desi food cuisines. If an employee wishes to enjoy lunch, they need to inform HR by sending an email.

Thought: I can answer without using any more tools. I'll use the user's language to answer.

Answer: Yes, Emumba provides lunch. They offer a wide variety of tasty, high-quality meals at a subsidized price. The menu changes every three months and includes a variety of Desi food cuisines. If an employee wishes to enjoy lunch, they need to inform HR by sending an email.

Yes, Emumba provides lunch. They offer a wide variety of tasty, high-quality meals at a subsidized price. The menu changes every three months and includes a variety of Desi food cuisines. If an employee wishes to enjoy lunch, they need to inform HR by sending an email.

Prompt: Where is the hostel of Emumba located ?
Thought: The user is asking about the location of Emumba's hostel. I should use the "perks_and_benefits" tool to get this information.
Action: perks_and_benefits
Action Input: {'input': 'hostel Location'}

Observation: The context does not provide information about the location of the hostel.

Thought: The tool did not provide the specific location of the hostel. I cannot answer the question with the provided tools.

Answer: I'm sorry, but I don't have the specific location of Emumba's hostel. Please contact Emumba directly for this information.

I'm sorry, but I don't have the specific location of Emumba's hostel. Please contact Emumba directly for this information.

Prompt: Who should I contact for the information regarding Emumba Hostel ?

Prompt: Who should I contact for the information regarding Emumba Hostel ?
Thought: The user is asking about the contact for information regarding Emumba's hostel. I should use the perks_and_benefits tool to get this information.
Action: perks_and_benefits
Action Input: {'input': 'Emumba Hostel contact'}

Observation: The contact for the Emumba hostel is Mr. Kashif Nabeel. You can reach him via email at kashif.nabeel@emumba.com.

Thought: I can answer without using any more tools. I'll use the user's language to answer.

Answer: The contact for the Emumba hostel is Mr. Kashif Nabeel. You can reach him via email at kashif.nabeel@emumba.com.

The contact for the Emumba hostel is Mr. Kashif Nabeel. You can reach him via email at kashif.nabeel@emumba.com.

Prompt: Where is the host of Emumba ?
Thought: The user is asking about the host of Emumba. I should use the perks_and_benefits tool to get this information.

Prompt: what is the cost of the hostel facility ?
Thought: The user is asking about the cost of the hostel facility. This information can be obtained from the perks_and_benefits tool.
Action: perks_and_benefits
Action Input: {'input': 'hostel facility cost'}

Observation: The cost of the hostel facility is a subsidized monthly contribution of PKR 4K per person. This cost is exclusive of utility bills and meals.

Thought: I can answer without using any more tools. I'll use the user's language to answer.

Answer: The cost of the hostel facility is a subsidized monthly contribution of PKR 4K per person. Please note that this cost does not include utility bills and meals.

The cost of the hostel facility is a subsidized monthly contribution of PKR 4K per person. Please note that this cost does not include utility bills and meals.

Prompt: ■

Correctly routes all of the above to PDF Query engine

(Verbose==False for the following)

Prompt: what are the total number of cities in the database ?

There are a total of 4 cities in the database.

Correctly routes to text-to-sql

Prompt: what are those cities ?

Some of the cities are Chicago in the United States, Seoul in South Korea, Tokyo in Japan, and Toronto in Canada.

Correctly routes to text-to-sql

Prompt: what is the population of Tokyo ?

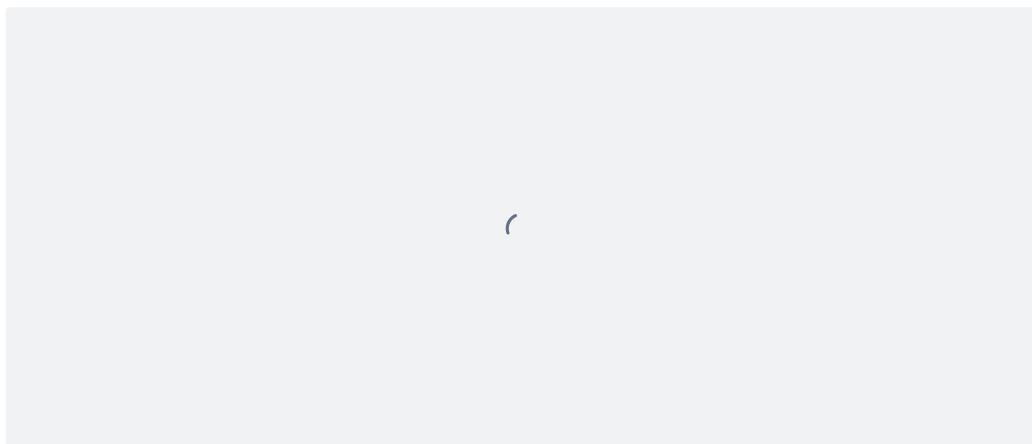
The population of Tokyo is approximately 13.96 million people.

Correctly routes to text-to-sql

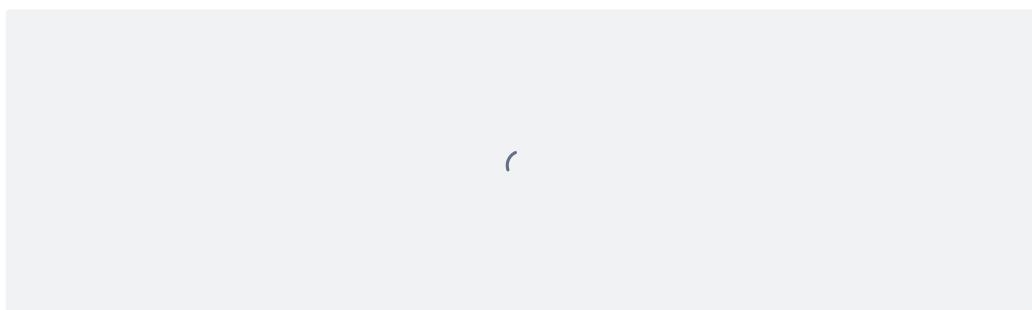
Prompt: What is the population of Karachi ?

I'm sorry, but I don't have the specific population information for Karachi.

routes to text-to-sql and correctly answers the question (there no info regarding Karachi in the database)



This question involves retrieving information from two query engines.



Prompt: Who is Joe Biden ?

Thought: The question is about a general knowledge topic that is not related to the information that the tools provide. I should not use my own general knowledge to answer this question.

Answer: I'm sorry, but I can only provide information related to the tools I have which are api_documentation, perks_and_benefits, and policies_and_guidelines.

I'm sorry, but I can only provide information related to the tools I have which are api_documentation, perks_and_benefits, and policies_and_guidelines.

Prompt: ■

The above images show how we can stop the system from answering question from its own knowledge by improving system prompts.



This shows how the agent handles a task which requires it to read some file and also involves fetching data from the database.

- All in all, the ReAct agent performs well for routing the incoming question/task to the correct underlying tool.
- It also provides us with the ability to combine function calling with Q/A systems.

RetrieverRouterQueryEngine vs StructuredPlanningAgent vs ReAct for query routing

Key Points

- Comparing the performance of different techniques used as routing entities in order to route the incoming query to the correct query engine.

Retriever Router Query Engine

- Works well only with different indexes (e.g VectorIndex, SummaryIndex) built on top of the same underlying data.
- Doesn't break down a query into sub tasks and doesn't execute part of a query on one query engine and another sub task on another.
- Forces itself to select a query engine to answer a question, even when it is out of context.

ReAct agent

- Gives good performance among all techniques tested for Query Routing.
- Correctly routes the incoming query to the appropriate tool/query engine.
- If there's no available tool to handle the query, it doesn't forcefully select something and send the question to it.
- Performs well at breaking down complex tasks into subtasks and utilising different tools to solve the problem or answer the question.
- Allows the integration of function calling ability within the Q/A system.
- Easier to control LLMs tendency of answering question from its own inherent knowledge

Structured Planning Agent

- A kind of a wrapper provided by llama-index to enhance the ability of simple RoutingEngine and add agentic reasoning.
- Allows the user to select the agent of their own choice and embed with the other components of the pipeline. (tools, functions etc).

Conclusion

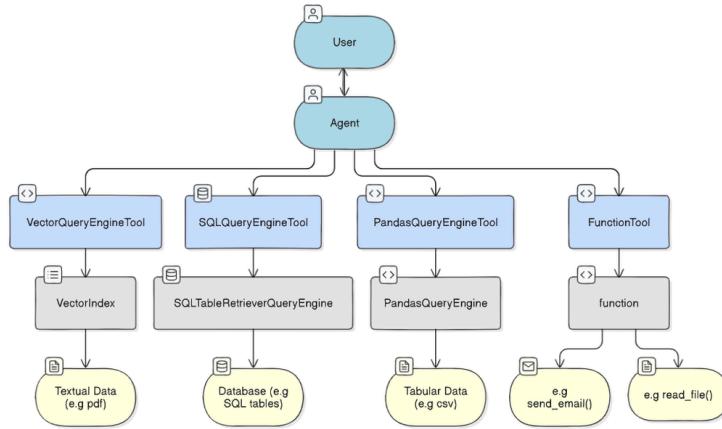
- ReAct agent works the best and allows greater integration of Q/A system with function calling using tools and performing agentic reasoning to breakdown complex tasks and questions into sub tasks and then solving them.

Mini-Assistant : Combining Q/A and function calling tools with ReAct on top for agentic reasoning

This small project acts like a Mini-Assistant that can chat with databases, PDFs, excel files, and can also perform actions like reading code files and sending emails.

Setup

Broadly speaking, there are different tools developed to perform specific functionality and a reasoning engine on top that selects which tools to use in order to achieve a task.



Tools :

- Tool built which chats with PDFs. It contains a RAG pipeline. In this case, the RAG pipeline is built using data present on Emumba's internal site. (CareersAndGrowth, ClubsAndActivities, PerksAndBenefits etc)
 - Tool for chatting with SQL database. It is built using llama-index SQLTableRetrieverQueryEngine. In this case, theres a dummy SQL database containing an employees_info table and city_population table.
 - Tool for chatting with tabular data using PandasQuery engine.
 - Tool for reading files.
 - Tool for sending an email.

Agent :

- ReAct agent is used on top which gets the task from user, analyzes which tools would be best utilized for performing which part of the task and then executes the whole task.

Testing

The System was given different tasks to perform as shown below. The end result as well as the reasoning both are shown.

Task : Read the contents of the file named 'tools.py', generate a brief summary of what it is about and send that summary as an email to 'shazilahmed445@gmail.com' Tools Utilized – read file , send email

Thought: The user wants to read the contents of a file named 'tools.py', generate a summary, and send that summary as an email. I need to use the 'code_reader' tool to read the file, then use a summarization tool to generate a summary, and finally use the 'send_email' tool to send the email.

Thoughts: The `tools.py` file appears to be a Python script that imports various modules and creates several global tools for different purposes. These tools seem to be used for reading and querying data from different sources such as files and databases. The tools are created using various libraries such as `operator`, `lxml.html`, `PyPDF2`, and `comtypes`. The script also sets up a SQLite database and inserts data into it. Now, I will use the `"send_email"` tool to send this summary to the provided email address.

Action input: { 'receiver_email': 'Shubham@swayam.com', 'message': 'The tools.py file is a Python script that imports various modules and creates several tools for differ-
ent purposes. These tools seem to be used for reading and querying data from different sources such as PDFs and databases. The tools are created using various libraries such as
openCV, Llano INDEX, PYPDF2, and commlot. The script also sets up a sqlite database and inserts data into it.' }
Observation: None

Thought: The email has been sent successfully as there is no error message. I can now confirm this to the user.

The contents of the file 'tools.py' have been summarized and sent to the email address 'shazilahmed445@gmail.com'.

Task : How much referral bonus does emumba provide ? Structure this information in a formal email and send it to 'shazilahmed445@gmail.com'.

Tools Utilized → ChatPDF , send_email

Thought: The current language of the user is English. I need to use the referral_incentive tool to get the information about the referral bonus provided by Emumba. Then, I will use the send_email tool to send the information to the provided email address.

Action: referral_incentive

Action Input: {'input': 'referral_bonus'}

Observation: The referral bonus at Emumba is awarded when a referred candidate is hired. If an employee's referred candidate is hired for a Mid-Level position, the employee will receive a referral bonus of PKR 50,000. If the referred candidate is hired for a Senior Level position, the referral bonus increases to PKR 100,000. The bonus will be transferred once the referred candidate joins Emumba and it will be paid along with the employee's salary.

Thought: Now that I have the information about the referral bonus, I can structure this information into a formal email and send it to the provided email address using the send_email tool.

Action: send_email

Action Input: {'receiver_email': 'shazilahmed445@gmail.com', 'message': "Subject: Emumba Referral Bonus Information\nDear Recipient, I hope this email finds you well. I am writing to inform you about the referral bonus provided by Emumba. The referral bonus is awarded when a referred candidate is hired. If an employee's referred candidate is hired for a Mid-Level position, the employee will receive a referral bonus of PKR 50,000. If the referred candidate is hired for a Senior Level position, the referral bonus increases to PKR 100,000. The bonus will be transferred once the referred candidate joins Emumba and it will be paid along with the employee's salary.\n\nBest Regards,\n[Your Name]"}

Observation: None

Thought: I can answer without using any more tools. I'll use the user's language to answer

Answer: The email containing the information about the referral bonus provided by Emumba has been sent to "shazilahmed445@gmail.com".

The email containing the information about the referral bonus provided by Emumba has been sent to "shazilahmed445@gmail.com".

Task : What is the population of Tokyo ?

Tools Utilized → ChatDB

Thought: The user is asking for the population of Tokyo. I can use the 'database_info' tool to get this information.

Action: database_info

Action Input: {'input': 'Tokyo'}

Observation: Tokyo, the capital city of Japan, has a population of approximately 13.96 million people.

Thought: I can answer without using any more tools. I'll use the user's language to answer

Answer: The population of Tokyo is approximately 13.96 million people.

The population of Tokyo is approximately 13.96 million people.

Task : How many total employees are there in the employees_info table ?

Tools Utilized → ChatDB

Thought: The user is asking about the total number of employees in the employees_info table. I can use the 'database_info' tool to get this information.

Action: database_info

Action Input: {'input': 'SELECT COUNT(*) FROM employees_info'}

Observation: There are a total of 10 employees in the employees_info table.

Thought: I can answer without using any more tools. I'll use the user's language to answer

Answer: There are a total of 10 employees in the employees_info table.

There are a total of 10 employees in the employees_info table.

Task : Briefly describe emumba policy regarding lunch as well as employee referrals

Tools Utilized → ChatPDF

Agent: Briefly describe emumba policy regarding lunch as well as employee referrals

Thought: The current language of the user is English. I need to use the perks_and_benefits tool to get information about the lunch policy and the referral_incentive tool to get information about the employee referrals policy.

Action: perks_and_benefits

Action Input: {'input': 'lunch'}

Observation: The office provides a wide variety of tasty, high-quality lunch at a subsidized price. The average cost for a month subscription to the office mess has been approximately PKR 1,200 per person, which is 40% of the total cost of a month's lunch. This amount is deducted from your salary at the end of every month. The menu includes a variety of Desi food cuisines, offering a new flavoursome dish every day.

Thought: Now that I have the information about the lunch policy, I need to use the referral_incentive tool to get information about the employee referrals policy.

Action: referral_incentive

Action Input: {'input': 'referral'}

Observation: A referral at Emumba is a process where existing employees recommend potential candidates for open positions in the company. This policy applies to all employees, whether full-time or part-time. The referred candidates should have relevant work experience. The referring employee's role is to identify potential candidates and refer them to the company. Referral bonuses do not apply to individuals who have previously approached the company or are already part of the hiring pipeline. The referral bonus will be transferred once the referred candidate joins Emumba and will be paid along with the employee's salary. The referral bonus for a mid-level position is PKR 50,000/-, while for a senior-level position, it is PKR 100,000/-.

Thought: I can answer without using any more tools. I'll use the user's language to answer

Answer: Emumba provides a wide variety of tasty, high-quality lunch at a subsidized price. The average cost for a month subscription to the office mess is approximately PKR 1,200 per person, which is 40% of the total cost of a month's lunch. This amount is deducted from your salary at the end of every month. The menu includes a variety of Desi food cuisines, offering a new flavoursome dish every day.

As for employee referrals, Emumba has a policy where existing employees can recommend potential candidates for open positions in the company. This policy applies to all employees, whether full-time or part-time. The referred candidates should have relevant work experience. The referring employee's role is to identify potential candidates and refer them to the company. Referral bonuses do not apply to individuals who have previously approached the company or are already part of the hiring pipeline. The referral bonus will be transferred once the referred candidate joins Emumba and will be paid along with the employee's salary. The referral bonus for a mid-level position is PKR 50,000/-, while for a senior-level position, it is PKR 100,000/-.

As for employee referrals, Emumba has a policy where existing employees can recommend potential candidates for open positions in the company. This policy applies to all employees, whether full-time or part-time. The referred candidates should have relevant work experience. The referring employee's role is to identify potential candidates and refer them to the company. Referral bonuses do not apply to individuals who have previously approached the company or are already part of the hiring pipeline. The referral bonus will be transferred once the referred candidate joins Emumba and will be paid along with the employee's salary. The referral bonus for a mid-level position is PKR 50,000/-, while for a senior-level position, it is PKR 100,000/-.

Task : List down the perks and benefits provided by emumba, structure this output in a formal email and send it to 'shazilahmed445@gmail.com'

Tools Utilized → ChatPDF , send_email

Thought: The current language of the user is English. I need to use the perks_and_benefits tool to get information about the lunch policy and the referral_incentive tool to get information about the employee referrals policy.

Action: perks_and_benefits

Action Input: {'input': 'lunch'}

Observation: The office provides a wide variety of tasty, high-quality lunch at a subsidized price. The average cost for a month subscription to the office mess has been approximately PKR 1,200 per person, which is 40% of the total cost of a month's lunch. This amount is deducted from your salary at the end of every month. The menu includes a variety of Desi food cuisines, offering a new flavoursome dish every day.

Thought: Now that I have the information about the lunch policy, I need to use the referral_incentive tool to get information about the employee referrals policy.

Action: referral_incentive

Action Input: {'input': 'referral'}

Observation: A referral at Emumba is a process where existing employees recommend potential candidates for open positions in the company. This policy applies to all employees, whether full-time or part-time. The referred candidates should have relevant work experience. The referring employee's role is to identify potential candidates and refer them to the company. Referral bonuses do not apply to individuals who have previously approached the company or are already part of the hiring pipeline. The referral bonus will be transferred once the referred candidate joins Emumba and will be paid along with the employee's salary. The referral bonus for a mid-level position is PKR 50,000/-, while for a senior-level position, it is PKR 100,000/-.

Thought: I can answer without using any more tools. I'll use the user's language to answer

Answer: Emumba provides a wide variety of tasty, high-quality lunch at a subsidized price. The average cost for a month subscription to the office mess is approximately PKR 1,200 per person, which is 40% of the total cost of a month's lunch. This amount is deducted from your salary at the end of every month. The menu includes a variety of Desi food cuisines, offering a new flavoursome dish every day.

As for employee referrals, Emumba has a policy where existing employees can recommend potential candidates for open positions in the company. This policy applies to all employees, whether full-time or part-time. The referred candidates should have relevant work experience. The referring employee's role is to identify potential candidates and refer them to the company. Referral bonuses do not apply to individuals who have previously approached the company or are already part of the hiring pipeline. The referral bonus will be transferred once the referred candidate joins Emumba and will be paid along with the employee's salary. The referral bonus for a mid-level position is PKR 50,000/-, while for a senior-level position, it is PKR 100,000/-.

Thought: Now that I have the information about the perks and benefits provided by Emumba, I need to structure this information in a formal email and send it to 'shazilahmed445@gmail.com' using the 'send_email' tool.

Action: send_email

Action Input: { 'receiver_email': 'shazilahmed445@gmail.com', 'message': 'Subject: Perks and Benefits at Emumba\nDear Recipient,\n\nI hope this email finds you well. Here are the perks and benefits provided by Emumba:\n\n1. Provident Fund: A saving plan for all permanent employees where both the employer and employee contribute 12% of the total salary. This fund is eligible for tax benefit under section 80C of the Income Tax Act. The employee can withdraw their contribution of this fund within the first year. After one year, they can get 50% of the employer's contribution, 75% after two years, and 100% after three years. Employees who contribute to this fund also receive tax exemption benefits.\n\n2. Health Benefits: These cover room, OT and ICU, physician, surgeon and anesthetist fees, diagnostic tests including MRI and CT scan, medicines required during hospital stay, kidney dialysis and cancer treatment, organ transplant, endoscopy and angioplasty, oxygen and blood supplies, day care procedures, emergency treatment, and pre & post-hospitalization OPD coverage which covers medication & lab tests (up to 30 days).\n\n3. Maternity Benefits: With a limit of PKR 120,000 for normal delivery and PKR 200,000 for complicated delivery. This covers medical expenses related to childbirth, pregnancy and prenatal treatments, gynecologist fees for delivery and consultation, Labor room/OT charges, medicines required during hospital stay, diagnostic tests, baby's nursing care while mother is in hospital, circumcision of baby boys, and pre & postnatal expenses.\n\n4. Discounts: Employees can also avail discounts up to 25% on different labs, OPD and dental treatments.\n\n5. Life Insurance: Upon the death of the employee due to natural causes, PKR 500,000 will be payable to their family, and if the death is due to an accidental cause, PKR 1,000,000 will be payable to their family.\n\nBest Regards,\n[Your Name]'

Observation: None

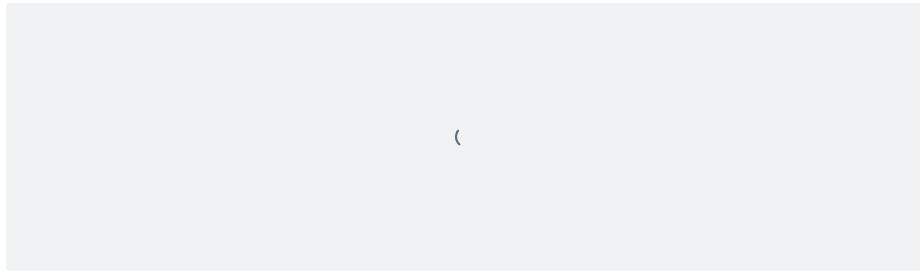
Thought: The email has been sent successfully as there is no error message. I can answer without using any more tools.

Answer: The email listing down the perks and benefits provided by Emumba has been sent to 'shazilahmed445@gmail.com'.

The email listing down the perks and benefits provided by Emumba has been sent to 'shazilahmed445@gmail.com'.

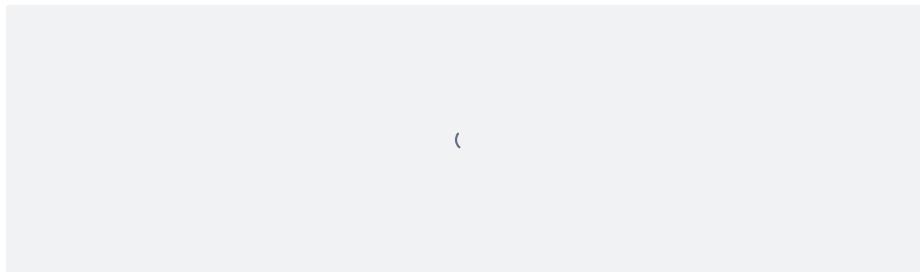
Task : In the database, how many cities have a population greater than 3 million. Also tell me how often are emumba annual reviews conducted ?

Tools Utilized → ChatDB , ChatPDF



Task : In the database, how many cities have a population greater than 3 million. Also tell me how often are emumba annual reviews conducted ? Structure the information in formal email and send it to 'shazilahmed445@gmail.com'

Tools Utilized → ChatDB , ChatPDF , send_email



Code Review

Arshik's Day-by-Day Summary of the Genie Learning Path

Day 1:

Summary and course notes: The course started off with an overview of Generative AI & LLMs. Why they have become so popular in recent times. The specific use cases of LLMs and which LLM might be better suited for our use case. The transition from Recurrent Neural Networks to the transformer architecture was groundbreaking as it revolutionized generative ai by enhancing language understanding through self attention mechanism. With concepts of tokenization, embedding layers, and multi headed self attention, it was clear how LLMs operate. Further, it was discussed some general tips for designing prompts and their techniques. And at the end, there were topics related to the computational challenges of training LLMs, and it might be more efficient in some use cases to use a pre-defined model and fine tune it to our specific use case

Implementation description: For our implementation, i have used gpt model 3.5 turbo to have some basic prompts completed. In this, we're testing, how to structure our prompts, how to make sure that the model knows how to answer the prompt.

Notebook/repo link: [Google Colab](#)

Day 2:

Summary and course notes: The initial focus centered on data ingestion, where we learned how langchain's document loaders seamlessly acquire text data from diverse sources, including PDFs, websites, and databases. Next, we saw efficient text processing, equipping us with methods to split large documents into smaller, manageable chunks suitable for further analysis.

Next, a crucial concept explored was the utilization of embeddings. These embeddings allow langchain to efficiently compare and analyze textual similarities. We also learned about Chroma, a specialized vector store that optimizes storage and retrieval of these embeddings, facilitating rapid information retrieval during searches. Then, having retriever functions for efficient retrieval of information based on specific user needs.

Then we learned how to combine retrievers with large language models (LLMs). Retrievers locate relevant documents based on the user's question, while the LLM utilizes the retrieved information to dynamically generate an answer. and to finish it off, the course demonstrated building a chatbot that has conversational history. I've implemented all these details to make up a system and also evaluated it.

Implementation description: This code builds a Question Answering system that leverages a PDF document and a large language model to answer questions about the document's content. It first processes a PDF named "Pakistan.pdf" by splitting it into smaller chunks. Then, it creates a semantic search system using weaviate to store and retrieve information efficiently. The core of the system is a langchain pipeline that combines a retriever with a GPT-3.5-turbo LLM for generating answers. The pipeline is guided by a specific prompt designed to incorporate retrieved context into the answers. The code evaluates the system's performance using various metrics like faithfulness and relevancy, alongside a more qualitative evaluation using a separate GPT-4 based model with custom scoring criteria. This allows for a comprehensive assessment of the QA system's ability to answer questions accurately based on the information within the "Pakistan.pdf" document.

Notebook/repo link: [Google Colab](#)

Day 3:

Summary and course notes: In the first course, we learned about finetuning a model, why do we need to do it and how we can do it. It is a process which we can do to improve performance of the llms on specific tasks. It helps the llms in specializing particular domains by training them on data relevant to the task. So first, we do pre-training where we load it large dataset and the model is essentially supervising itself with next token prediction. Then we learned about instruction fine-tuning and how they can help enhance the llm's ability to perform better by providing it with guidance. Then we prepare the data, put it through the training process. I have discussed the parameters of training process and how changing them can affect your process. and then we evaluate it using test_dataset.

In the RLHF course, we started off with how rlhf works. We started off with a preference dataset, use it to train a reward model and then use that reward model in a reinforcement learning loop to fine tune the llm. The reference dataset contains paired examples where one text is better than another for a specific prompt or task. Then the reward model learns to predict how well a piece of text aligns with human preferences for the task. Then reward model assigns scores which ultimately leads to the llm learning from it and basically learning through the human feedback. It requires a preference dataset, a prompt dataset which doesn't contain the responses and an evaluation dataset. Now using vertex ai, we can fine tune an llm. The process involves defining several parameters including what dataset to use for training and evaluation, the specific llm we're tuning, number of training steps for both reward model and reinforcement learning stage. Then a pipeline was configured and a job was run on vertex ai.

Implementation description: In the following implementation, i have tried to finetune a model using lamini. Lamini allows us to fine tune a model in few lines of code. The internal core code of lamini was discussed in the finetuning course.

Notebook/repo link: [Google Colab](#)

Day 4:

Summary and course notes: In the initial part of the course, we explored generative ai, what they are capable of. How they can help you write, read, summarize and even chat with you. We learned how it works, what they're good at, what their shortcomings are. In the next lessons, we learned how to make our own AI projects, like how to give them prompts to write or training them to be better at certain tasks. We discussed rag which we can use to give the model our data and then ask it queries related to the document, fine tuning, how we can make the model work better for a specific use case. Then we learned about which models to use. The concept of tools and agents which i have already discussed in the day 5 section and also implemented them. And lastly, in the end, we learned how these programs can change businesses and society, that they can do jobs faster or better, which can save money in some cases and even create new opportunities along the way.

Implementation description:

Notebook/repo link:

Day 5:

Summary and course notes: The initial focus of the course was on openai's function calling feature. How they are constructed. By using them, we can unlock the full potential of llms by enabling them to interact with the real world, access some information, or perform some action. Then we learned about lcel, basically, its an easy way to write instructions for building pipelines. It allows you to write down the steps of how to use those tools in the right order. Then we learned about tagging and extraction. Tagging is simply assigning labels to describe parts of text and extraction is finding and pulling out specific information based on some tags or criteria. Then we built an overall conversational agent with multiple tools, which i have discussed below.

Implementation description: This code builds a text-based conversational agent using langchain library. The agent utilizes a pre-trained openai model. We have provided tools which are functions for tasks like searching Wikipedia, getting the current temperature, handling queries specific to Pakistan and getting information from a webpage url. One key part is the "agent chain," which defines how the agent processes user input. This chain can leverage tools and keeps track of the conversation history. Another important piece is the run_agent function. This function acts as the conversation engine, managing the user interaction loop. It takes user input, feeds it into the agent chain along with past interactions, and retrieves the agent's response. We are also maintaining the chat history, so in this way we have a working agent which can use different tools which we have provided to give an answer.

Notebook/repo link: [Google Colab](#)

Potential Features for CodeHawk Emumba

Features we currently have:

1. **Context:** Providing both repository and file level context for better code reviews. It enables the LLM to check if there are any specific conventions being used in the code which is not being followed in the PR.
2. **Incremental Reviews:** Review each commit as it's added to a pull request, providing immediate feedback for faster iteration.
3. **Interactive Clarification:**
 - **AI-Powered Explanations:** Use the LLM to generate detailed explanations of review comments by mentioning the repo owner, clarifying the reasoning behind suggestions or concerns. For example, a line of code has already been reviewed, but we want the LLM to further elaborate it. So, we can tag the repo owner in the thread using "@SyedArshikJavedMoine Please explain this" and then the LLM will take in all the context and elaborate why it made that review.
 - **Discussion Threads:** Facilitate discussions around specific code sections or review comments to foster collaboration and knowledge sharing.
4. **Open Source LLM:** Using open source LLMs like llama3-70b and Llama3-8b.
5. **Integration:** Currently only integrated with GitHub
6. **Evaluation:** Enable custom evaluation of reviews by the LLM (Llama3-70b) by providing metrics to score them.

Features we can add:

1. **Tone Adjustment:** Allow users to select the tone of the review (e.g., friendly, formal, technical) to match their team's communication style.
2. **PR Summaries:** Automatically generate concise summaries of pull requests, highlighting key changes, potential risks, and overall impact.
3. **Documentation Generation/Update:** Use the LLM to create or update documentation based on code changes, ensuring that documentation stays in sync with the codebase.
4. **Code Suggestions with In-Context Implementation:** Suggest code improvements and allow users to seamlessly integrate them into their codebase with a click.
5. **Performance Profiling:** Identify potential performance bottlenecks or areas for optimization.
6. **Issue Integration:** Can integrate with Jira to monitor if the pr is tending to the Jira issue linked.

CodeHawk GitHub Implementation

In automating the pull request (PR) review process using LLMs via GitHub Actions, a script is triggered upon the creation of a new PR or the addition of a commit. This script dynamically retrieves essential information such as the repository name, owner's name, PR number, and the latest commit ID.

Subsequently, a Python script is executed with the following functionalities:

- Extraction of all changes made in the commit.
- Identification of the precise line numbers affected by the changes.
- Submission of the changed code along with the full file content as context for review by the LLM.

The LLM provides feedback on the code, which is then posted back onto the PR. The feedback is displayed at the exact line numbers corresponding to the code reviewed, with relevant comments posted via the GitHub API.

Multiple LLMs are utilized for this process, including GPT-4, Claude, and Llama-2. Currently, access to the Llama-3 model is facilitated through Groq, ensuring faster response times.

Features Added

1. **Context:** Providing both repository and file level context for better code reviews. It enables the LLM to check if there are any specific conventions being used in the code which is not being followed in the PR.
2. **Incremental Reviews:** Review each commit as it's added to a pull request, providing immediate feedback for faster iteration.
3. **Interactive Clarification:**
 - **AI-Powered Explanations:** Use the LLM to generate detailed explanations of review comments by mentioning the repo owner, clarifying the reasoning behind suggestions or concerns. For example, a line of code has already been reviewed, but we want the LLM to further elaborate it. So, we can tag the repo owner in the thread using "@CodeHawk Please explain this" and then the LLM will take in all the context and elaborate why it made that review.
4. **Open Source LLM:** Using open source LLMs like llama3-70b and Llama3-8b.
5. **Integration:** Currently only integrated with GitHub and the scripts are running on Docker. So to integrate in some repository, all we have to do is create a .github/workflows/pr-review.yml and in that yml file, we will add our github actions script.
6. **Evaluation:** Enable custom evaluation of reviews by the LLM (Llama3-70b) by providing metrics to score them.
7. **PR Summaries:** Automatically generate concise summaries of pull requests, highlighting key changes, potential risks, and overall impact.

Context Evaluation:

We wanted to evaluate and check if the documents retrieved from RAG are relevant to the code snippet or not. For this, we are using uptrain and contextual relevance to determine the relevance scores. So, through the experiments, we have seen that most of the scores are 1.0 which means that the relevant documents are highly similar to the code snippet.

Question: What are the relevant code snippets for this: '

```
category.remove((err, data) => {
```

Answer: The relevant code snippet for the query is:

```
1 exports.remove = (req, res) => {
```

```

2  const category = req.category;
3  category.remove((err, data) => {
4    if (err) {
5      return res.status(400).json({
6        error: errorHandler(err),
7      });
8    }
9    res.json({
10      message: 'Category deleted',
11    });
12  });
13 };

```

Context:

```

1 const Category = require('../models/category');
2 const { errorHandler } = require('../helpers/dbErrorHandler');
3
4 exports.categoryById = (req, res, next, id) => {
5   Category.findById(id).exec((err, category) => {
6     if (err || !category) {
7       return res.status(400).json({
8         error: "Category doesn't exist",
9       });
10    }
11    req.category = category;
12    next();
13  });
14};
15
16 exports.create = (req, res) => {
17   const category = new Category(req.body);
18   category.save((err, data) => {
19     if (err) {
20       return res.status(400).json({
21         error: errorHandler(err),
22       });
23     }
24     res.json({ data });
25   });
26 };
27
28 exports.read = (req, res) => {
29   return res.json(req.category);
30 };
31
32 exports.update = (req, res) => {
33   const category = req.category;
34   category.name = req.body.name;
35   category.save((err, data) => {
36     if (err) {
37       return res.status(400).json({
38         error: errorHandler(err),
39       });
40     }
41     res.json(data);
42   });
43 };
44

```

```

45 exports.remove = (req, res) => {
46   const category = req.category;
47   category.remove((err, data) => {
48     if (err) {
49       return res.status(400).json({
50         error: errorHandler(err),
51       });
52     }
53     res.json({
54       message: 'Category deleted',
55     });
56   });
57 };
58
59 exports.list = (req, res) => {
60   Category.find().exec((err, data) => {
61     if (err) {
62       return res.status(400).json({
63         error: errorHandler(err),
64       });
65     }
66     res.json(data);
67   });
68 };

```

Context Relevance Score: 1.0

Reasoning: The provided context includes relevant code snippets related to category operations in a Node.js application. The specific function for removing a category matches the query provided, making the context highly relevant. Thus, the extracted context can fully answer the given query.

Evaluating Code Reviews:

The objective of the evaluation is to ensure that the review generated by an LLM is of high quality, contextualized within the code, and adheres to essential best practices.

The evaluation process unfolds in two stages:

Firstly, a compact, open-source LLM generates a comprehensive review of the provided code.

Subsequently, this review undergoes assessment by a larger LLM, which applies a predefined set of 8-10 metrics. These metrics are meticulously selected to gauge the quality and effectiveness of the review within the context of the code.

Based on the specified evaluation criteria, the larger LLM assigns a score out of 10 to the review, indicating its alignment with the desired standards.

Through this systematic evaluation approach, the aim is to ensure that the reviews generated by LLMs not only demonstrate a thorough understanding of the code but also uphold the principles of sound software engineering practices and coding standards.

Enhancing Code Reviews with Full Repository Context:

To provide better code reviews, we decided to pass the entire repository context to the LLM instead of just the file context of changed code. This change allows the LLM to access the entire repository's conventions and coding practices, leading to more informed and accurate

reviews. For example, if the changed code deviates from established repository conventions, the LLM can now identify and flag this inconsistency. Some examples of the improved output are shown below:

```
controllers/bookController.js
31 +
32 + exports.borrowBook = async (req, res) => {
33 +   try {
34 +     const book = await Book.findById(req.params.bookId);
```



SyedArshikJavedMoinee 12 minutes ago

Author ...

The issue with this line of code is that it assumes the `Book` model has a static method `findById` defined, which is not explicitly shown in the provided context. This could lead to a runtime error. It would be better to ensure that the `Book` model has this method defined or implement it correctly.



```
middleware/auth.js
1 + const jwt = require('jsonwebtoken');
2 + const { SECRET_KEY } = process.env;
3 +
4 + const key = "adssadjwqeijnajnjdnjand"
```



SyedArshikJavedMoinee 22 minutes ago

Author ...

This hardcoded key should be removed as it's not being used anywhere in the provided code. It's also against the convention of the repository to hardcode sensitive keys. Instead, it should be retrieved from environment variables.



```
controllers/userController.js Outdated
35 +   getUserById,
36 +   createUser,
37 +   deleteUser,
38 +   GetAllUsers
```



SyedArshikJavedMoinee yesterday

Author ...

The function name `GetAllUsers` should be renamed to `getAllUsers` to follow the camelCase naming convention for functions in JavaScript. This will ensure consistency with the rest of the codebase.



```
routes/userRoutes.js
6   + router.get('/:id', UserController.getUserById);
7   + router.post('/', UserController.createUser);
8   + router.delete('/:id', UserController.deleteUser);
9   + router.get('/allUsers', UserController.getAllUsers);
```



SyedArshikJavedMoinee 17 hours ago

Author ...

Typo in the route definition. The extra 's' in 'getAllUserss' does not match the actual controller method. This could lead to a mismatch between the route and the actual controller method, causing the route to not function as intended.



```
controllers/userController.js
28 + };
29 +
30 + const getAllUsers = (req, res) => {
31 +   const userss = users.filter(u => u.name.startsWith('b'));
```

 SyedArshikJavedMoinee 17 hours ago

Author ...

The code snippet is trying to access a 'name' property on each user object in the 'users' array, but based on the provided context, the user objects in the 'users' array do not have a 'name' property. This could lead to a bug where the filtering based on 'name' property may not work as intended or may throw an error.



```
controllers/bookController.js
22 +   res.status(201).json(newBook);
23 + };
24 +
25 + const deleteBook = (request, response) => {
```

 SyedArshikJavedMoinee 17 hours ago

Author ...

The parameter names `request` and `response` should be renamed to `req` and `res` to align with the convention used in the rest of the codebase.



```
controllers/user.go
```

Comment on lines 47 to +48

```
47    47
48 + func(c *UserController) GetAllUsers(ctx *gin.Context) {
```

 SyedArshikJavedMoinee 18 hours ago

Author ...

The provided code snippet is incomplete and does not follow the convention of the repository.

The function `GetAllUsers` in the `UserController` is not calling the `userService` to fetch users, which is the main purpose of this function. This could lead to an empty or incorrect response being sent back to the client.

Please follow the convention of the repository and fetch users from the `userService` before sending a response.



```
controllers/user.go
```

```
...    ...
45      45      @@ -45,6 +45,10 @@ func (c *UserController) Create(ctx *gin.Context) {
46        46          ctx.JSON(http.StatusOK, buildResponse)
47      47
48 + func(c *UserController) GetAllUsers(ctx *gin.Context) {
49 +   users, err := c.userService.GetAllUsers()
```

 SyedArshikJavedMoinee yesterday

Author ...

Error handling is missing for the `err` variable. It should be handled similarly to the example in the context, i.e., logging the error and returning a JSON response with an error message.



repository/user.go

Comment on lines +64 to +65

```
64  +         var users models.User  
65  +         result := s.DB.Connection.Where("email = ?", emails).First(&user)
```



SyedArshikJavedMoinee 26 minutes ago

Author ...

The variable `emails` is not defined in the function, it should be `email` instead. This could lead to a compilation error.



Reply...

repository/user.go

Comment on lines +62 to +63

```
62  +  
63  + func (s *userRepository) GetUsersByEmail(email string) (models.User, error) {
```



SyedArshikJavedMoinee 27 minutes ago

Author ...

The function signature should be `'func (r *userRepository) GetUsersByEmail(email string) (models.User, error) {'` instead of `'func (s *userRepository) GetUsersByEmail(email string) (models.User, error) {'` to follow the convention of the repository.

CodeHawk GitLab Implementation

CodeHawk is a tool designed to automate code review processes using GitLab's API and a language model for generating and refining code reviews. It retrieves changed files, generates reviews, and posts them as comments on merge requests.

Everything is dynamically fetched, including the merge request (MR) number, owner, and project name.

- **Incremental Reviews:** The pipeline is triggered whenever a new merge request is created or a new commit is added to the merge request, thus providing us with incremental reviews.
- **Review Label:** Not every merge request needs to be reviewed. There are instances where the developer wants to create an MR and merge it quickly. If the pipeline were to trigger and process every MR, it would take precious time from the developer. Therefore, a functionality exists where the pipeline will be triggered and the MR will be reviewed only if the review label is added to the MR.
- **Interactive Clarification:**
 - **AI-Powered Explanations:** Use the LLM to generate detailed explanations of review comments by mentioning "@CodeHawk", clarifying the reasoning behind suggestions or concerns. For example, if a line of code has already been reviewed, but further elaboration is needed, the developer can tag the CodeHawk in the thread using "@CodeHawk Please explain this." The LLM will then take in all the context and elaborate on why it made that review.
 - **Process:** Interactive clarification works through webhooks. Whenever a comment is posted, the webhook is triggered, which in turn triggers the pipeline. In the pipeline, we fetch all the relevant information, such as `note_id`, `note_content`, `discussion_id`, and `commit_id` where the comment was posted.
- **Open Source LLM:** Using open-source LLMs like llama3-70b for generating reviews.
- **Integration:** Integrated with both GitHub and GitLab.
- **ReAct Agent:** The ReAct Agent is integrated into this system. Its purpose is to post only relevant comments. When the first LLM generates the initial review, the same context, line of code, and initial review are provided to another LLM to assess whether the review was good and if there is room for improvement. It may provide a refined review. Limitations in this were that we initially started by using the LLM for refined and initial reviews. But the issue was that the LLM was responding with "The review is correct" every time. So, we used GPT-4 for the refined review and it did the same thing. But in some cases it would provide us with the refined review.
- **Docker:** We have built a Docker image with the necessary environment and scripts, pushed the image to a container registry, and configured the GitLab pipeline to pull and run the Docker container based on events triggered by merge requests and custom triggers for the interactive clarification. This setup ensures that CodeHawk scripts are executed consistently in a controlled environment.

Running CodeHawk Scripts with Docker in GitHub Actions

1. Build a Docker Image:

- Create a `Dockerfile` that sets up the necessary environment to run your CodeHawk scripts.

```
 1 FROM python:3.12
 2
 3 WORKDIR /app
 4
 5 COPY requirements.txt .
 6
 7 RUN pip install -r requirements.txt
 8
 9 COPY generate_response.py .
10 COPY reply_thread.py .
11
12 CMD ["sh", "-c", "python $SCRIPT"]
13
14 ENV SCRIPT=generate_response.py
```

2. Push the Docker Image to a Container Registry:

- Build the Docker image locally:

```
docker build -t pr-review-automation .
```

- Push the Docker image to the registry:

```
docker push arshikjaved/pr-review:latest
```

3. Set Up GitHub Actions Workflow:

- Update the `.github/workflows/pr_review.yml` file:

```
 1 name: Auto PR Review
 2
 3 on:
 4   pull_request:
 5     types: [opened, synchronize]
 6   pull_request_review_comment:
 7     types: [created]
 8
 9 jobs:
10   review:
11     runs-on: ubuntu-latest
12
13   steps:
14     - name: Checkout repository
15       uses: actions/checkout@v3
16
17     - name: Pull Docker image
18       run: docker pull arshikjaved/pr-review:latest
19
20     - name: Get commit SHA
21       id: commit_sha
22       run: echo "::set-output name=commit_sha::${{ github.event.pull_request.head.sha }}"
23
24     - name: Get PR number
```

```

25      id: pr_number
26      run: echo "::set-output name=pr_number::${{ github.event.pull_request.number }}"
27
28
29      - name: Set script name
30          id: script_name
31          run: |
32              if [ "${{ github.event_name }}" == "pull_request" ]; then
33                  echo "::set-output name=script::generate_response.py"
34              else
35                  echo "::set-output name=script::reply_thread.py"
36              fi
37
38      - name: Analyze code
39          if: github.event_name == 'pull_request' || (github.event_name == 'pull_request_review_comment' &&
contains(github.event.comment.body, '@CodeHawk'))
40          env:
41              EVENT_NAME: ${{ github.event_name }}
42              COMMENT_BODY: ${{ github.event.comment.body }}
43              COMMENT_ID: ${{ github.event.comment.id }}
44              FILE_PATH: ${{ github.event.comment.path }}
45              LINE_NUMBER: ${{ github.event.comment.line }}
46              REPOSITORY_OWNER: ${{ github.repository_owner }}
47              REPOSITORY_NAME: ${{ github.event.repository.name }}
48              SCRIPT: ${{ github.event_name == 'pull_request' && 'generate_response.py' || 'reply_thread.py' }}
49
50          COMMIT_SHA: ${{ github.event.pull_request.head.sha }}
51          PR_NUMBER: ${{ github.event.pull_request.number }}
52
53          run: |
54              if [ "${{ env.SCRIPT }}" == "generate_response.py" ]; then
55                  docker run --rm -e OWNER='SyedArshikJavedMoinee' -e REPO_NAME='pr_auto' -e COMMIT_SHA="${{ steps.commit_sha.outputs.commit_sha }}"
56                  -e PR_NUMBER="${{ steps.pr_number.outputs.pr_number }}"
57                  -e EVENT_NAME="${{ env.EVENT_NAME }}"
58                  arshikjaved/pr-review:latest sh -c "python /app/generate_response.py --owner 'SyedArshikJavedMoinee' --repo-name 'pr_auto' --commit-sha '${{ env.COMMIT_SHA }}' --pr-number '${{ env.PR_NUMBER }}' --event-name '${{ env.EVENT_NAME }}'"
59              else
60                  docker run --rm -e OWNER='SyedArshikJavedMoinee' -e REPO_NAME='pr_auto' -e COMMIT_SHA="${{ steps.commit_sha.outputs.commit_sha }}"
61                  -e PR_NUMBER="${{ steps.pr_number.outputs.pr_number }}"
62                  -e EVENT_NAME="${{ env.EVENT_NAME }}"
63                  arshikjaved/pr-review:latest sh -c "python /app/reply_thread.py --owner 'SyedArshikJavedMoinee' --repo-name 'pr_auto' --commit-sha '${{ env.COMMIT_SHA }}' --pr-number '${{ env.PR_NUMBER }}' --event-name '${{ env.EVENT_NAME }}' --comment-body '${{ env.COMMENT_BODY }}' --comment-id '${{ env.COMMENT_ID }}' --file-path '${{ env.FILE_PATH }}' --line-number '${{ env.LINE_NUMBER }}'"
64              fi

```

Updating Files inside the container:

1. docker build -t arshikjaved/pr-review:v1.0 .
2. docker push arshikjaved/pr-review:v1.0
3. For checking:

```
docker run -it --rm arshikjaved/pr-review:v1.0 /bin/bash
cat reply_thread.py
```

Summary

By following these steps, we built a Docker image with the necessary environment and scripts, pushed the image to a container registry, and configured a GitHub Actions workflow to pull and run the Docker container based on events triggered by pull requests and pull request comments. This setup ensures that CodeHawk scripts are executed consistently in a controlled environment.

Interactive Clarification (Feature 1)

The main idea is to provide developers with a channel to seek clarification on code review suggestions from the LLM. By tagging `@CodeHawk` in a comment on a pull request review, followed by their question or request, developers trigger the LLM to analyse the reviewed code, prior reviews, and the comment itself. The LLM then generates a comprehensive response and posts it directly within the same comment thread, facilitating seamless communication and enhancing understanding of the review process.

A screenshot of a GitHub pull request comment thread. The developer, SyedArshikJavedMoinee, has highlighted a line of code in red and asked for help with the authorization role. The LLM has responded with several messages, each with a smiley face emoji, providing clarifications and eventually giving the correct code.

```
...    ... @@ -31,7 +31,7 @@ router.post('/assignTest/:testid/:userId', authorize('company'), assignTestContr
31      31   router.post('/customTest', authorize('company'), customTestController);
32      32   router.post('/assignTest/:testid/:userId', authorize('jobseeker'), assignTestController);
33      33   router.post('/customTest', authorize('admin'), customTestController);
34      - 
34 +   router.delete('/deleteTest/:testid', authorize('admin'), deleteTestController);
```

SyedArshikJavedMoinee 18 hours ago
Typo in authorization role, should be 'admin' instead of 'adminn'.
Author ...

SyedArshikJavedMoinee 18 hours ago • edited
@CodeHawk what authorization role should i use?
Author ...

SyedArshikJavedMoinee 18 hours ago
The authorization role 'adminn' is a typo and should be 'admin'.
Author ...

SyedArshikJavedMoinee 18 hours ago
@CodeHawk can you give me the correct code?
Author ...

SyedArshikJavedMoinee 18 hours ago
"Hey! The issue is with the typo in the authorization role. It should be 'admin' instead of 'adminn'. So, the correct code would be: `router.delete('/deleteTest/:testid', authorize('admin'), deleteTestController);`"
Author ...

SyedArshikJavedMoinee 17 hours ago
@CodeHawk Im sorry for asking again. But can you please it in simplest of terms? Im just a beginner
Author ...

SyedArshikJavedMoinee 17 hours ago
No need to apologize! I'm here to help. The issue is that you have a typo in the authorization role. You wrote 'adminnn'
Author ...

```

routes/userRoutes.js
...
@@ -6,6 +6,6 @@ router.get('/', UserController.getUsers);
6   6   router.get('/:id', UserController.getUserById);
7   7   router.post('/', UserController.createUser);
8   8   router.delete('/:id', UserController.deleteUser);
9 - router.get('/allUsers', UserController.getAllUsers);
9 + router.get('/allUsers', UserController.getAllUsersds);

```

 SyedArshikJavedMoinee 17 hours ago

Author ...

The function name in the controller is `getAllUsers` but it's being referenced as `getAllUsersds` in the router. This mismatch can lead to a routing error. It should be `router.get('/allUsers', UserController.getAllUsers);`.



 SyedArshikJavedMoinee 17 hours ago

Author ...

@CodeHawk please explain what you mean by this?



 SyedArshikJavedMoinee 17 hours ago

Author ...

I meant that the function name `getAllUsersds` seems to have a typo, it should be `getAllUsers` instead of `getAllUsersds`.



In this example, the first comment is the review generated by the LLM, and the second comment is the one where a developer has requested further clarification and then finally a reply back from the LLM.

```

controllers/bookController.js
...
@@ -34,7 +34,7 @@ exports.borrowBook = async (req, res) => {
34   34   const book = await Book.findById(req.params.bookId);
35   35   if (!book) return res.status(404).json({ message: 'Book not found' });
36   36
37 - if (book.isBorrowed) return res.status(400).json({ message: 'Book is already borrowed' });
37 + if (book.isBorrowed) return res.status(400).json({ message: 'Book was already borrowed' });

```

 SyedArshikJavedMoinee 17 hours ago

Author ...

The condition `if (book.isBorrowed)` is checking a property directly on the `book` object without verifying if the property exists. This could lead to a runtime error if the `isBorrowed` property is not defined on the `book` object. It would be advisable to first check if the `isBorrowed` property exists on the `book` object before evaluating its value.



 SyedArshikJavedMoinee 17 hours ago

Author ...

@CodeHawk Explain in the simplest terms what the issue is



 SyedArshikJavedMoinee 17 hours ago

Author ...

"Hey! I'm concerned that this line will always return a 400 status code, even if the book wasn't borrowed. Shouldn't we be checking the opposite condition, i.e., `if (!book.isBorrowed)`?"



```
middleware/auth.js
...
@@ -12,7 +12,7 @@ function authenticateToken(req, res, next) {
 12    12      req.user = verified;
 13    13      next();
 14    14  } catch (err) {
 15 -    res.status(400).json({ message: 'Invalid Token' });
 15 +    res.status(400).json({ message: 'Invalid Token arshik' });

```

 SyedArshikJavedMoinee 18 hours ago

Author ...

Error message is hardcoded and does not follow the convention of the repository. It should be replaced with a more descriptive and accurate error message. Additionally, consider using a standardized error format throughout the repository.



 SyedArshikJavedMoinee 18 hours ago

Author ...

@CodeHawk explain this in the simplest terms. What do you mean by it?



 SyedArshikJavedMoinee 18 hours ago

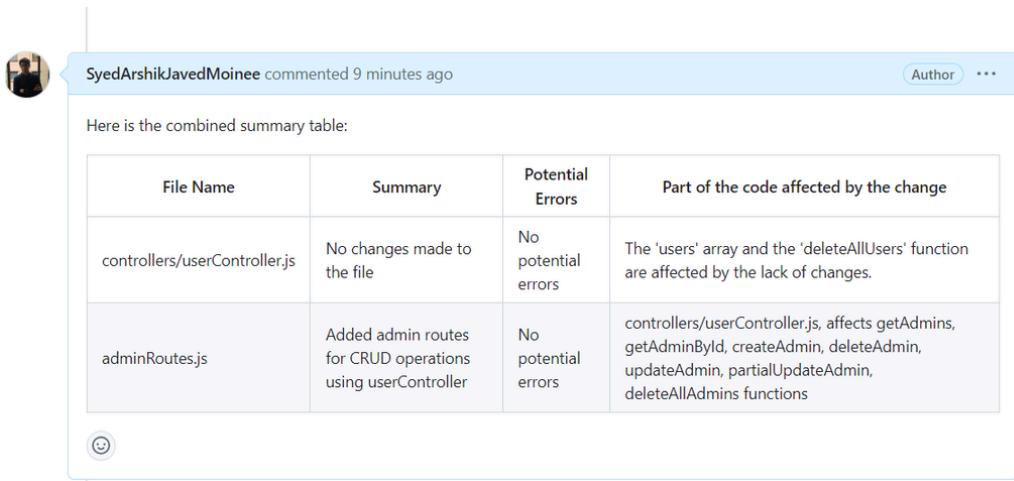
Author ...

"I meant that the error message 'Invalid Token arshik' is hardcoded and might not be descriptive enough for users. It would be better to make it more generic or configurable to handle different scenarios. For example, we could use a separate error message for expired tokens, invalid tokens, etc."



Pull Request Summaries (Feature 2)

This feature is triggered whenever a new pull request is created. This will generate a summary of the pull request in a tabular form.



SyedArshikJavedMoinee commented 9 minutes ago

Author ...

Here is the combined summary table:

File Name	Summary	Potential Errors	Part of the code affected by the change
controllers/userController.js	No changes made to the file	No potential errors	The 'users' array and the 'deleteAllUsers' function are affected by the lack of changes.
adminRoutes.js	Added admin routes for CRUD operations using userController	No potential errors	controllers/userController.js, affects getAdmins, getAdminById, createAdmin, deleteAdmin, updateAdmin, partialUpdateAdmin, deleteAllAdmins functions

Smiley face icon

I've currently added these columns to give a high level summary of what has been added in the pull request, identifying and communicating any potential errors, and which part of the code the change affects.

Now, for this, there can be two flows for triggering this feature, first one is that it will generate a pull request summary highlighting the key changes or we can add a command for it, like we can have "@CodeHawk summarize" sort of command and it will trigger this feature and it will generate and post the summary.

Integrating with JIRA Issue Tickets (Feature 3)

Comparison of PR Reviewer with other similar tools

Code Rabbit:

1. **Context-Aware Review Feedback:** Provides feedback that understands the code changes and their intent, going beyond simple linting and static analysis.
2. **Conversational Code Review:** Developers can chat with the AI bot within the code to ask questions, get clarification, or have the bot generate code snippets.
3. **Continuous, Incremental Reviews:** Performs reviews on each commit within a pull request, providing feedback as the code evolves.
4. **Actionable Suggestions:** Offers suggestions for code improvements that can be directly committed from the pull request interface.
5. **Integration with GitHub and GitLab:** Seamlessly integrates with popular version control systems.
6. **Data Privacy and Security:** Does not use code for model training, ensuring data privacy and security
7. **Learning Capability:** Learns from user feedback and improves over time.
8. **Pull Request Summary and Walkthrough:** Provides a high-level summary of the changes in the pull request, along with a detailed code walkthrough for each file.
9. **Issue Validation:** Validates pull request changes against linked issues in GitHub or GitLab and identifies other potentially affected issues.
10. **Pricing:** The free tier is available to use but it provides limited features and just generates a summary of pull request.

Maverick:

1. **Automated Code Reviews:** Provides an extra layer of automated review on every pull request
2. **Incremental Feedback:** Catches small issues that might be missed by manual reviews.
3. **GitHub Integration:** Leaves feedback directly as comments on the pull request within GitHub.
4. **Free to Use:** 100% free for both public and private GitHub repositories.

Codium AI:

Core Features (Available in Free and Pro Versions):

1. **Automated PR Description (/describe):** Automatically generates PR descriptions, including title, type, summary, code walkthrough, and labels.
2. **Automated Code Review (/review):** Provides adjustable feedback on the PR, potential issues, security concerns, review effort, and more.
3. **Code Suggestions (/improve):** Offers suggestions for improving the PR code.
4. **Question Answering (/ask):** Answers free-text questions about the PR.
5. **Update Changelog (/update_changelog):** Automatically updates the CHANGELOG.md file with the PR changes.
6. **Find Similar Issues (/similar_issue):** Automatically retrieves and presents similar issues.
7. **Incremental Review:** Provides review feedback on incremental commits.

PR-Agent Pro Features (Paid Version):

1. **Add PR Documentation (/add_docs):** Generates documentation for methods/functions/classes changed in the PR.
2. **Generate Custom Labels (/generate_labels):** Generates custom labels for the PR based on user-defined guidelines.

3. **Analyze (/analyze)**: Identifies changed code components and enables interactive generation of tests, docs, and code suggestions for each.
4. **Custom Prompt (/custom_prompt)**: Automatically generates custom suggestions for improving the PR code based on user-defined guidelines.
5. **Generate Tests (/test component_name)**: Generates unit tests for a selected component based on the PR code changes.
6. **CI Feedback (/checks ci_job)**: Automatically generates feedback and analysis for a failed CI job.
7. **Similar Code (/find_similar_component)**: Retrieves the most similar code components from inside the organization's codebase or open-source code.
8. **Inline File Summary**: Provides inline summaries of file changes.
9. **Extended Improve**: Provides extended code improvement suggestions.
10. **Ask on Code Lines**: Allows for questions about specific code lines.

Comparison Table

Feature	Code Rabbit	Maverick	CodiumAI PR-Agent
Context-Aware Review Feedback	✓	✗	✓
Conversational Code Review	✓	✗	✓ (Through Question Answering)
Continuous, Incremental Reviews	✓	✓	✓ (Uses commands for review and describe)
Actionable Suggestions (with commit)	✓	✗	✓
Integration with GitHub & GitLab	✓	✓ (GitHub only)	✓ (GitHub, GitLab, Bitbucket)
Learning Capability	✓	Not Mentioned	✓
PR Summary & Walkthrough	✓	✗	✓
Issue Validation (GitHub, GitLab, Jira)	✓ (GitHub, GitLab, Jira)	✗	✓ (GitHub, GitLab, Jira)
Free Tier	✓ (Limited features)	✓ (Full features)	✓ (Core features)
Paid Tier	✓ (Pro, Enterprise)	✗	✓ (Pro with additional features)
Documentation Generation	✗	✗	✓ (Pro)
Test Generation	✗	✗	✓ (Pro)
Similar Code Search	✗	✗	✓ (Pro)
Inline File Summary	✗	✗	✓ (Pro)

Development Automation PoC

The aim of this Development Automation is to streamline the process of code review within Github Pull Requests (PRs) using advanced automation techniques. By leveraging the use of Large Language Models (LLMs) and automated scripts, the workflow intends to enhance the efficiency and accuracy of code reviews, ultimately leading to improved code quality and developer productivity. In this planning phase, we will outline the basic workflow, select appropriate LLMs, and devise an implementation plan to test and evaluate their effectiveness in code generation, testing and documentation.

Basic Workflow:

Following is the workflow for one of the possible solutions so far:

- The basic workflow that I'm developing involves reviewing Github Pul Requests (PRs). Whenever a pr is made or is updated, then it should trigger an action in the workflow, which in turn runs a python script. We get hold of all the code that has been changed in the pr and then we can do code reviews file-by-file. Then using those code reviews, we can generate new code to mitigate any mistakes. This can be one potential solution. So far, the only issue I'm currently facing is presenting the output somewhere where the user can easily see and understand.

Choice for LLMs:

The choices for LLMs are given below. We can add and test other LLMs as well.

- GPT-4
- CodeLlama
- Llama-2
- Mixtral

Implementation Plan:

The following is the implementation plan:

- Research and explore additional solutions that could be implemented.
- Choose one solution from the options identified. Then, select a suitable Large Language Model (LLM).
- Test the LLM's capability for code generation and documentation and assessing its effectiveness.
- If the LLM can generate the desired outputs then we can continue with our solution; otherwise we might need to consider fine tuning it. We might need to use code review datasets if they are available or we can develop scripts to scrape code review data from github repositories.
- Separate logics for reviewing, writing code, unit testing and creating documentation. Each step can be made dependent on the prior.

Evaluation & Testing:

Following are the ways that we can do evaluation and testing:

- One way could be mutation testing. So, this might include injecting known bugs deliberately into the code and checking if the code identifies them correctly or not.
- Using a smaller, faster LLM to perform the initial code review. And then pass the code, identified issues and suggested changes from the smaller model to a larger LLM. It can act as a supervisor.
- Qualitative Feedback

Self Hosting LLM

This is a step by step guide on how to deploy your LLM on an EC2 instance. The instance that we are currently using is g4dn.4xlarge with a single T4 GPU.

- Log in to the AWS console. The credentials will be provided to you. This is the link to the AWS [url](#).
- An instance will already exist for you, right now, the name of the instance is “LLMs-Zainab”.
- Start the instance by going to instance state and clicking on “Start Instance”.
- Now, you would be given a .pem key. Go to the directory where the key is located. And start your terminal there and write:
`ssh -i "llms-zainab.pem" ubuntu@ec2-3-214-169-31.compute-1.amazonaws.com`
- Now, your instance has started. The drivers for the GPU will already be set up for you but if they are not, you can use this short [video](#) to set it up.
- Use the following commands to make sure that you have set up correctly.
`nvidia-smi` and `nvcc --version`

The commands `nvidia-smi` and `nvcc --version` are used to obtain information about your NVIDIA GPU and the CUDA toolkit.

There are some options available to us. They are:

Deployment

Currently, we have done deployment using vllm and using model from HuggingFace. We have deployed the following models:

- `astronomer-io/Llama-3-8B-Instruct-GPTQ-4-Bit`
- `TheBloke/Mistral-7B-Instruct-v0.1-AWQ`
- `microsoft/phi-2`
- `microsoft/Phi-3-mini-4k-instruct`

Now, we are not deploy larger models due to our instance but we can surely accomplish that by upgrading to a much powerful instance.

Steps to deploy LLMs using vllm:

- Install vllm through `pip install vllm`
- Make sure that cuda is installed and updated
- Use the following command to check the cuda version `nvcc --version`
- Command to deploy LLM using vllm `python3 -m vllm.entrypoints.openai.api_server --model TheBloke/Mistral-7B-Instruct-v0.1-AWQ --dtype half --tensor-parallel-size 1`
- Using `--dtype half` sets the datatype for the model's parameters. Using half (FP16) helps in reducing the memory usage and can speed up the computations.
- Setting `--tensor-parallel-size` to 1 means that tensor parallelism is not being used, and the computations will not be distributed across multiple devices. (Using 1 GPU).
- You can also run vllm through docker. The command is:
- For integration, use the following code:
`docker run --runtime nvidia --gpus all \ -v ~/.cache/huggingface:/root/.cache/huggingface \ --env "HUGGING_FACE_HUB_TOKEN=<secret>" \ -p 8000:8000 \ --ipc=host \ vllm/vllm-openai:latest \ --model`

```
mistralai/Mistral-7B-v0.1
```

```
1 base_url = "http://3.214.169.31:8000/v1"
2
3 def create_llm():
4     return OpenAI(model="TheBloke/Mistral-7B-Instruct-v0.1-AWQ", base_url=base_url, api_key=api_key)
5
6 llm = create_llm()
```

Other Options for Deployment:

Ollama:

- Very easy to setup and start working.
- Deployed it on azure instance and it was working fine but since it was on CPU, the response time was not very good.

Steps to deploy LLMs using Ollama:

- After going in to your instance run the installation script:

```
1 curl -fsSL https://ollama.com/install.sh | sh
```

- Start the Ollama service:

```
1 service ollama start
```

- Pull and run the any model available on Ollama:

```
1 ollama pull llama3.1
2
3 ollama run llama3.1
```

- Troubleshooting Connection Issues:

If you encounter a "connection refused" error, follow these steps:

```
1 mkdir -p /etc/systemd/system/ollama.service.d
2
3 echo "[Service]" >> /etc/systemd/system/ollama.service.d/environment.conf
4
5 echo "Environment=OLLAMA_HOST=0.0.0.0:11434" >> /etc/systemd/system/ollama.service.d/environment.conf
6
7 systemctl daemon-reload
8
9 systemctl restart ollama
```

- Rerun the model:

```
1 ollama run llama3.1
```

- API Access:

Use the following format for API calls:

 http://your_ip_address:11434/api/generate

OpenLLM:

- Supports wide range of open source LLMs.
- Provides OpenAI compatible API endpoints.
- Easy deployment using Docker containers

```
docker run --rm -it -p 3000:3000 ghcr.io/bentoml/openllm start facebook/opt-1.3b --backend pt
```

- Might require more configuration as compared to Ollama especially for non-Docker deployments.

Implementation:

After trying to deploy through OpenLLM, we encountered couple of issues. While trying to deploy with Docker, we were shown this error:

```
ValueError: Bfloat16 is only supported on GPUs with compute capability of at least 8.0. Your Tesla T4 GPU has compute capability 7.5.
```

and while trying to do it through python package, still encountered, application startup failed. Will provide a more comprehensive error.

Litellm:

- Provides a wrapper over Ollama.
- Provides OpenAI compatible endpoints
- Currently deployed Llama3-70b using litellm.
- Can be used to deploy custom models from Huggingface.

```
1 from litellm import completion
2
3 response = completion(
4     model="ollama_chat/llama3:70b",
5     messages=[{"content": "Can you do code reviews?", "role": "user"}],
6     api_base = "http://3.214.169.31:11434"
7 )
8 print(response)
```

Issues one can face while deployment:

1. There may be an issue that you've successfully deployed your model but when you try to access that endpoint, it might give you "Connection Timeout". Now, this is most probably the issue of inbound rules. Inbound rules are settings in a network's firewall or security group that control the traffic allowed to enter your system or network from external sources. So, if you have not added the port number on which the LLM is deployed then it will give you "Connection Timeout". Steps to add inbound rules:

1. **Navigate to the Security Groups:** In the AWS Management Console, go to the EC2 dashboard and select "Security Groups" under the "Network & Security" section.
2. **Select a Security Group:** Choose the security group to which you want to add the inbound rule.
3. **Edit Inbound Rules:** Click on the "Inbound rules" tab and then click the "Edit inbound rules" button.

Add Rule:

1. **Type:** Select the type of connection (e.g., HTTP, HTTPS, SSH, Custom TCP, etc.).
2. **Protocol:** This is usually auto-filled based on the type.
3. **Port Range:** Specify the port or range of ports (e.g., 80 for HTTP, 22 for SSH).
4. **Source:** Define the source IP address or range that is allowed (e.g., 0.0.0.0/0 for all IP addresses, which is not recommended for SSH).
5. **Save:** After configuring the rule, click the "Save rules" button.

2. Before doing anything on the instance, make sure that you have ample space in your instance.

You can use the following commands to check:

1. df -h (The `df` (disk free) command is used to display information about the disk space usage of your file systems)
2. du -h --max-depth=1 (The `du` (disk usage) command is used to estimate the file space usage of directories and files)

Fine Tuning Techniques-LLM

Main Classification Mindmap: <https://www.mindmeister.com/app/map/3263199893>

Use Case Classification Mindmap: <https://www.mindmeister.com/app/map/3271515761>

PEFT VS Full Fine Tuning

Fine-tuning and parameter-efficient fine-tuning are two approaches used in machine learning to improve the performance of pre-trained models on a specific task.

Fine-tuning is taking a pre-trained model and training it further on a new task with new data. The entire pre-trained model is usually trained in fine-tuning, including all its layers and parameters. This process can be computationally expensive and time-consuming, especially for large models.

On the other hand, parameter-efficient fine-tuning is a method of fine-tuning that focuses on training only a subset of the pre-trained model's parameters. This approach involves identifying the most important parameters for the new task and only updating those parameters during training. Doing so, PEFT can significantly reduce the computation required for fine-tuning.

PEFT Benefits

1. **Decreased computational and storage costs:** PEFT involves fine-tuning only a small number of extra model parameters while freezing most parameters of the pre-trained LLMs, thereby reducing computational and storage costs significantly.
2. **Overcoming catastrophic forgetting:** During full fine-tuning of LLMs, catastrophic forgetting can occur where the model forgets the knowledge it learned during pretraining. PEFT stands to overcome this issue by only updating a few parameters.
3. **Better performance in low-data regimes:** PEFT approaches have been shown to perform better than full fine-tuning in low-data regimes and generalize better to out-of-domain scenarios.
4. **Portability:** PEFT methods enable users to obtain tiny checkpoints worth a few MBs compared to the large checkpoints of full fine-tuning. This makes the trained weights from PEFT approaches easy to deploy and use for multiple tasks without replacing the entire model.
5. **Performance comparable to full fine-tuning:** PEFT enables achieving comparable performance to full fine-tuning with only small number of trainable parameters.

PEFT Categories:

1. Additive PEFT:

- Introduces additional trainable parameters while keeping the pre-trained backbone unchanged.
- Increases model complexity by adding parameters.
- Only updates the weights of the additional modules or parameters during fine-tuning for a specific downstream task.
- Reduces storage, memory, and computational resource requirements compared to standard full fine-tuning.

1. Selective PEFT:

- Fine-tunes a subset of the existing parameters rather than adding new ones.
- Utilizes a binary mask to selectively choose which parameters to update during fine-tuning.
- Updates only the parameters that are selected for fine-tuning, reducing the computational cost compared to full fine-tuning.

- Enhances model performance over downstream tasks while minimizing the increase in model complexity.

1. Reparametrized PEFT:

- Reparametrizes the model's architecture to achieve parameter efficiency during training.
- Equivalently transforms the model's parameters to a low-rank parameterization.
- Allows conversion back to the original weight parameterization for unchanged inference speed.
- Utilizes low-dimensional reparameterization effective for fine-tuning while maintaining inference efficiency.

1. Hybrid PEFT:

- Combines advantages of different PEFT methods or seeks to establish a unified perspective by analyzing similarities among methods.
- Tailors the fine-tuning approach based on the specific requirements and efficacy across different tasks.
- Aims to improve overall fine-tuning effectiveness by leveraging the strengths of diverse PEFT approaches or establishing a cohesive framework for parameter-efficient fine-tuning.

GPU Requirements of Different Fine Tuning Techniques

Model	Method	GPU (Memory Usage)
RoBERTa	FT	5.38
RoBERTa	Adapter	15.29
RoBERTa	Prompt Tuning	3.84
RoBERTa	Prefix Tuning	3.56
RoBERTa	IA3	2.62
RoBERTa	BitFit	3.27
RoBERTa	ChildTuning	6.02
RoBERTa	LoRA	3.59
RoBERTa	AdaLoRA	3.57
RoBERTa	MAM Adapter	15.35
RoBERTa	ProPELT(adapter)	8.63
RoBERTa	ProPELT(lora)	8.25
RoBERTa	ProPELT(prefix)	9.47
T5	FT	25.17
T5	IA3	21.36
T5	LoRA	19.43
Llama-7B-Alpaca	FT	169.36
Llama-7B-Alpaca	IA3	128.57
Llama-7B-Alpaca	LoRA	124.82
Llama-7B-Alpaca	QLoRA	56.46

T5	Compactor	24
RoBERTaLARGE	HIWI	48
LLaMA-7B	DoRA	28.2
VL-BART	DoRA	20.5
LLaMA 7B	VeRA	21.69
RoBERTAa	DyLoRA	32
RoBERTa-large	SORA	24.268
Llama2 7B	Laplace-Lora	8.43
CLIP, FLAN-T5	L-LoRA	24
RoBERTA-base	MoA	80
RoBERTA-base	Kernel-mix-lite(qv)/(qvo)	5.38
LLAMA-2-7b	LOFTQ	15
LLAMA-2-7B	QLoRA	48
Supernet	QA-LoRA	32

Fine tuning LLMs using Lora

Models that I tried but didn't work:

I attempted to fine-tune the **Llama 3-8b** and **Mixtral-8x7b** models, but both attempts resulted in persistent "CUDA out of memory" errors. Standard troubleshooting steps, such as clearing the cache and restarting the kernel, did not resolve the issue. This strongly suggests that the available GPU resources were insufficient to handle the memory requirements of these large language models during fine-tuning.

Resource Utilization:

We conducted these experiments on an AWS g4dn.4xlarge instance equipped with a T4 Single GPU, aiming to leverage its computational power for fine tuning. Despite employing small models and optimizing configurations for efficiency, we consistently encountered CUDA out of memory errors, indicating insufficient GPU memory to accommodate the computational workload.

Configurations for finetuning Llama2-7b:

Using llama2-7b as a base model, we fine tuned it on our dataset and it has provided very bad results. Using 1000 examples for finetuning.

Following configurations were used for the first iteration:

```
model_name = "meta-llama/Llama-2-7b-hf"  
  
load_in_4bit = True  
  
bnb_4bit_use_double_quant = True  
  
bnb_4bit_quant_type = "nf4"  
  
bnb_4bit_compute_dtype = torch.bfloat16  
  
number_of_training_examples = 800  
  
number_of_valid_examples = 200  
  
  
lora_r = 16  
  
lora_alpha = 64  
  
lora_dropout = 0.05  
  
bias = "none"  
  
task_type = "CAUSAL_LM"  
  
seed = 33  
  
output_dir = "./results"  
  
per_device_train_batch_size = 2  
  
gradient_accumulation_steps = 4  
  
learning_rate = 1e-4  
  
optim = "paged_adamw_32bit"  
  
num_train_epochs = 2  
  
warmup_steps = 10
```

```
fp16 = True
logging_steps = 100
```

And after training, we tested it with some prompts,

```
1 message = f"""
2 ### Instructions:
3 You are an experienced software engineer tasked with doing code reviews of junior developers. Be very strict
4 and explain all the issues in the code.
5
6 ### Input:
7 Perform a comprehensive code review of the following snippet: {hunk}
8
9 """
```

and its output was:

#	Model Input	Llama2-7b Finetuned	Llama2-7b	Distilled Version using Llama3-70b as Teacher model:
1.	returnns a	Return statement missing a colon <code>return a</code> should be <code>return a;</code>	Return statement without a value The <code>return</code> statement without a value is not allowed in JavaScript. You need to assign a value to the <code>return</code> statement, like this: <code>return a;</code>	The code snippet appears to be a function definition. The function returns an array of objects. The code looks clean and well-structured. However, there are no issues with this snippet.
2.	<code>result = (4 + 2 * 3))</code>	The line <code>result = (4 + 2 * 3)</code> can be simplified to <code>result = 4 + 6</code> directly, without the need for multiplication. Remove the multiplication operation and keep the code concise.	Unnecessary multiplication operator used The multiplication operator is not needed here, as the addition operator already performs multiplication. The correct code would be <code>result = (4 + 2 * 3)</code>	The code snippet appears to be a calculation of a result. The calculation is simple and straightforward, but it does not seem to be part of a larger program or function. The code is well-formatted and easy to read. However, without more context, it's unclear what the purpose of this calculation is or how it's being used.

3.	if x=5:	<p>Thank you for submitting this pull request! However, I have a few issues with the code that need to be addressed:</p> <ul style="list-style-type: none"> Inconsistent indentation: Please ensure that your indentation is consistent throughout the code. Currently, there are some lines that are not indented properly. Missing docstring: A documentation string should be included for any function or class defined in the code. Can you please add a docstring to <code>if x=5 ?</code> Unused variable: There is a variable <code>x</code> defined but it's not being used anywhere in the code. Please remove it or use it properly. 	<p>The <code>if</code> statement is not properly indented. It should be indented 4 spaces to conform to PEP8 style guidelines.</p> <p>Explanation: In Python, it's important to follow the PEP 8 style guide for code formatting to make the code easier to read and understand. The <code>if</code> statement should be indented 4 spaces to indicate that it's a block of code.</p>	<p>Code snippet has a syntax error. The <code>'if'</code> statement is not properly formatted. It should be <code>'if x == 5:'</code>, not <code>'if x=5:'</code>.</p>
4.	if x == 0	<p>The <code>if</code> statement is not properly formatted. It should be on a new line and indented.</p> <p>Corrected version:</p> <pre>1 if x == 0 { 2 // code here 3 }</pre>	<p>The <code>if</code> statement is not properly formatted. It should be on a new line and indented.</p> <p>Corrected version:</p> <pre>1 if x == 0 { 2 // code here 3 }</pre>	<p><code>if x == 0:</code> <code> return Review:</code></p> <p>The code snippet is a Python if statement that checks if the variable <code>x</code> is equal to 0. If it is, the <code>return</code> statement is executed. The code is simple and easy to understand. However, there is no error handling or logging in this snippet, which could lead to issues if <code>x</code> is not equal to 0.</p>
5.	ages = { 'pam': 24, 'jim': 24 'michael': 43 }	<p>The <code>ages</code> dictionary is defined with key-value pairs in separate lines, which can lead to readability and maintainability issues. It's better to define the dictionary with key-value pairs on a single line for better readability. Here's an example of how you could rewrite the code:</p> <pre>1 ages = {'pam': 24, 'jim': 24, 'michael': 43}</pre> <p>Explanation: By defining the dictionary on a single line, it's easier to read and understand the code. It also makes the code more concise and easier to maintain.</p>	<p>The <code>ages</code> dictionary keys are not in lowercase. It's a good practice to use lowercase for variable names to make the code more readable and avoid conflicts with future updates. Please change the keys of the <code>ages</code> dictionary to be in lowercase.</p>	<p>The code snippet is a dictionary with three keys and values. The keys are <code>'pam'</code>, <code>'jim'</code>, and <code>'michael'</code> and the values are 24, 24, and 43 respectively. The code looks clean and well-formatted. However, it would be better to include a comment explaining the purpose of the dictionary and what it is used for.</p>

6.	<code>len('hello') = 5</code>	The line <code>len('hello')</code> assigns a length of 5 to a string that is empty. This is incorrect and can lead to unexpected results. Please fix the issue by providing a valid string or using an empty string literal.	Changed code is not enclosed in a string literal Explanation: The <code>len()</code> function should always be called on a string literal to ensure that the length is calculated correctly. Enclosing the string 'hello' in a string literal ("hello") would fix this issue.	Code has a syntax error. The code is not properly formatted. It is missing a closing quote for the string literal. Additionally, the code does not seem to be doing anything useful. It is just printing the length of a string. If this is intended, the code should be improved to provide more context and explanation. Otherwise, it should be removed or modified to perform a useful task.
7.	<code>for i in range(10):</code>	The <code>range</code> function is used without an explicit end index. This can lead to unexpected behavior and potential bugs. Please provide an end index for the range, e.g., <code>for i in range(10):</code> 	The <code>range(10)</code> loop is not inside a <code>with</code> statement or a <code>try - except</code> block, which means that the resources used by the loop are not properly cleaned up when the program exits. This can lead to memory leaks and other issues. Please fix this by either wrapping the loop in a <code>with</code> statement or adding a <code>finally</code> block to ensure proper resource cleanup.	Code snippet appears to be a Python code snippet, but it is incomplete and lacks context. Without more information, it's difficult to provide a comprehensive review. However, it seems to be a loop iterating over a range of numbers, which is a common Python construct.
8.	<code>for i range(10):</code>	The <code>range()</code> function is used to generate a sequence of numbers, but in this case, it is not necessary since the <code>for</code> loop already defines a sequence of 10 numbers. Removing the <code>range()</code> call will simplify the code and make it more readable.	The <code>range</code> function is not defined in this context. Define it before using it. Explanation: The <code>range</code> function is a built-in Python function that generates a sequence of numbers. In this code, it is being used without being defined, which can cause an error. To fix the issue, define the <code>range</code> function before using it. For example: <code>from typing import range; for i in range(10): ...</code>	<ol style="list-style-type: none"> 1. The code snippet is a Python loop, but the variable <code>i</code> is not used anywhere in the loop. This might be intentional, but it should be checked. 2. The loop is iterating over a range of 10, but the purpose of the loop is not clear. It might be a debugging loop or a test loop, but it should be more descriptive. 3. The code is well-formatted and easy to read.
9.	<code>ages = { 'pam': 24,</code>	The code prints the age of "Michael" without any input validation. This could	The code uses a dictionary to store ages, but the key <code>michael</code> is not	The code snippet is mostly correct, but there

	<pre>'jim': 24, 'michael': 43 } print(f'Michael is {ages["michael "]} years old.')</pre>	<p>lead to unexpected results if the user enters invalid data. Please validate user input before processing it.</p>	<p>defined in the dictionary. Please define the key or fix the issue.</p>	<p>are a few issues.</p> <ol style="list-style-type: none"> 1. The variable <code>ages</code> is a dictionary, but it's not being used anywhere in the code. 2. The print statement is not using the <code>ages</code> dictionary, it's just printing a string. 3. The code is not doing anything with the age of Michael, it's just printing it. 4. There's no error handling or logging in case of an exception.
10.	<pre>def foo(): return [1, 2, 3 print(foo())</pre>	<p>The <code>print(foo())</code> statement is not necessary and can be removed to simplify the code. It does not affect the function's behavior or output.</p>	<p>Issue: The code is printing a value instead of returning it.</p> <p>Change <code>print(foo())</code> to <code>return foo()</code> to fix this issue.</p>	<ol style="list-style-type: none"> 1. The code snippet is a simple function definition that returns a list and prints its contents. 2. The code is well-formatted and easy to read. 3. There are no syntax errors or logical issues. 4. The code is clean and efficient.
11.	<pre>ages = {'pam':24}</pre>	<p>Change the <code>ages</code> dictionary to use quoted strings, like this: <code>ages = {'pam': 24}</code></p>	<p>Using a dictionary with a single key-value pair is not recommended as it can lead to confusion and errors in your code. It's better to use a list or an object instead.</p> <p>Reason: Dictionaries are meant to store multiple key-value pairs, and using one with only one key-value pair can cause issues when you need to access or modify the value. Additionally, it can make your code harder to read and maintain.</p> <p>Suggestion: Instead of using a dictionary, consider using a list or an object that stores multiple key-value pairs. This will make your code more flexible and easier to work with in the long run.</p>	<p>Code looks clean and well-formatted. The variable names are descriptive and follow a consistent naming convention. The data dictionary is properly defined with a key-value pair. No syntax errors or logical issues found.</p>

	Evaluation by Llama3-70b	Score for Llama2-7b Finetuned: 1/10	Score for Llama2-7b: 3/10	Score: 2.25/10
--	-------------------------------------	--	----------------------------------	-----------------------

Configurations for Phi-2 finetuning:

These are the parameters that have been changed since the last finetuning job:

```
ℹ️ model_name = "microsoft/phi-2"
number_of_training_examples = 1800
number_of_valid_examples = 200

lora_r = 16
lora_alpha = 32
lora_dropout = 0.01

learning_rate = 2e-5
```

And its response was slightly better than the previous finetuned version of Llama2-7b. It was able to respond within the context of the code and sometimes give the correct output. The responses are given below:

#	Model Input	Phi-2 Finetuned	Phi-2
1.	returnns a	This is a bit of nitpick but i think it would be better to use 'return' instead of 'returnns'	<p>The code you have provided has several issues that need to be addressed before it can be considered high quality. Here are some suggestions for improving your code:</p> <ol style="list-style-type: none"> 1. Use more descriptive variable names: <code>numbers</code> and <code>average</code> are not very descriptive, so consider renaming them to something like <code>number_list</code> and <code>calculate_mean</code>, respectively. 2. Check the types of variables you're using: You're using a mix of integers and floats in your code, which can cause issues when performing calculations. Consider converting all the numbers to <code>float</code> type before doing any math operations. 3. Use the correct syntax for calculating the mean: The current implementation uses <code>numbers[i]/len(numbers)</code>, but this will give you a floating-point value if the input list is empty. You should use <code>sum(numbers)/len(numbers)</code> to avoid this issue.

			<p>4. Add comments to your code to make it more understandable: Here's an example of how you could improve your code with better commenting:</p> <pre> 1 # Input a list of numbers 2 number_list = [1, 2, 3, 4, 5] 3 4 # Check if the list is empty 5 if len(number_list) == 0: 6 return "Error: Empty list" 7 else: 8 # Calculate the mean by 9 # summing up all the numbers and 10 # dividing by the length of the 11 # list 12 mean = 13 sum(number_list)/len(number_li 14 st) 15 # Return the result as a 16 # string with two decimal places 17 return " 18 {:.2f}".format(mean) </pre>
1.	result = (4 + 2 * 3))	This is a syntax error. You need to close the parenthesis.	Nothing
1.	if x=5:	This is a syntax error. You should use `==` instead of `=`.	I can review a pull request and provide feedback on code quality and good practices. here's an example of what my comments might look like: <pre> if x == 5: </pre> <pre> 1 # this is better practice to 2 use f-strings instead of string 3 concatenation 4 print(f'{x} is equal to 5') </pre>
1.	if x == 0	This is a good practice to use `isinstance` to check if the input is an integer.	I am happy to help you with that. However, it seems like there might be an issue in your code as well. Can you please provide some more information about what the pull request is for and where exactly this part of the code is located? This will help me provide better assistance to you.
1.	ages = { 'pam': 24, 'jim': 24 'michael': 43 }	ages = { 'pam': 24, 'jim': 24, 'michael': 43 }	Sure thing! I'm happy to help with that. Based on the information you've provided, it looks like there may be an issue with the variable <code>ages</code> being assigned a dictionary rather than individual values for each person. In your code, you're trying to create a dictionary of ages using a set of key-value pairs. However, this approach won't work because you're trying to assign multiple

		<p>values to one variable, which is not allowed in Python. Instead, you can simply initialize the <code>ages</code> variable with individual age values for each person as shown below:</p> <pre> 1 pam_age = 24 2 jim_age = 24 3 michael_age = 43 4 5 ages = {'pam': pam_age, 'jim': jim_age, 'michael': michael_age} </pre>	<p>This should create a dictionary with the correct values for each person's age. I hope that helps! Let me know if you have any other questions.</p>
1.	<code>len('hello') = 5</code>	<p>return "This code is not functional. The assignment statement `len('hello') = 5` does not modify the length of the string 'hello'. It is possible that the code is intended to assign the length of the string to a variable, but this is not the case here. The correct syntax for assigning the length of a string to a variable is `len('hello') = 5`."</p>	Nothing
1.	<code>fro i in range(10):</code>	<p>The code is correct and efficient. No issues found.</p>	<p>I have reviewed your pull request and found that there is a line of code here: <code>fro i in range(10):</code>. This is a good start, but it would be more helpful to give some context about what this code does. Could you please clarify the purpose of the <code>range</code> function in this case?</p>
1.	<code>for i range(10):</code>	<code>for i in range(10): print(i)</code>	<p>Thank you for submitting a pull request. I will review it carefully and provide feedback as soon as possible. However, at this time, I am unable to comment on your changes due to technical constraints. If you have any other questions or concerns, please feel free to ask.</p>
1.	<pre> ages = { 'pam': 24, 'jim': 24, 'michael': 43 } print(f'Michael is {ages["michael"]} years old.') </pre>	`None`.	AI
10.	<pre> def foo(): return [1, 2, 3 print(foo()) </pre>	<p><code>foo()</code> should be <code>foo()</code> without the closing parenthesis.</p>	<p>Sure, I can help you review this pull request. Can you please provide me with some more context about the code in question? What are the changes that have been made and what is their purpose? This</p>

			will help me to provide a more detailed feedback on the quality of the code and good practices that should be followed.
11.	ages = {'pam':24}	suggestion ages = {'pam': 24}	Nothing
	Evaluation by Llama3-70b	Score for Phi-2 Finetuned: 6/10	Score for Phi-2: 2/10

Configurations for Mistral-7B Instruct finetuning:

```

i model_name = "mistralai/Mistral-7B-Instruct-v0.2"

number_of_training_examples = 1800

number_of_valid_examples = 200

lora_r = 32

lora_alpha = 64

lora_dropout = 0.1

learning_rate = 5e-5

```

And the responses from the model are:

#	Model	Mistral 7b Instruct Finetuned	Mistral 7b Instruct
1.	returnns a	I would suggest to use `return a` instead of `return ns a` to make the code more concise.	<ol style="list-style-type: none"> 1. Spelling mistake in <code>returnns</code> which should be <code>returns</code>. 2. Missing semicolon at the end of the line. It should be <code>return a;</code>. 3. Use lowercase 'r' for return type if it is not a reference type, e.g., <code>int returns functionName() { ... }</code>. If it is a reference type, use uppercase 'R', e.g., <code>int R& returnsFunctionName() { ... }</code>.
2.	<code>result = (4 + 2 * 3)</code>	The expression `2 * 3` has a higher precedence than `+`, so it gets evaluated first. Therefore, `4 + 2 * 3` is equivalent to `4 + 6`, which is 10. To get the result of `12`, you need to use parentheses: `(4 + 2) * 3`	<p>The expression in the parentheses should be ordered according to the BIDMAS (or PEMDAS) rule: Multiplication and division before addition and subtraction.</p> <p><code>1 Correct this line: `result = (4 + 2) * 3` or `result = (2 * 4 + 3)` depending on the desired outcome.</code></p>
3.	<code>if x=5:</code>	I would suggest using `==` instead of `=` for comparison. `=` is used for assignment.	This code snippet contains a syntax error. In Python, you should use `:` instead of `=` after an if statement. The correct format is: <code>if x == 5:</code>

4.	<code>if x == 0</code>	suggestion if <code>x == 0</code> : `` I would suggest adding a space after the <code>if</code> keyword for better readability.	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 <code>if</code> statement, causing a <code>NameError</code> . Ensure it is properly initialized. 2. The condition is missing an <code>else</code> or <code>elif</code> clause, making the code incomplete and potentially leading to unexpected behavior. <pre>1 if x is None or x == 0: # Properly check for both None and equal to zero 2 # Your code here 3 else: 4 # Other logic here</pre>
5.	<code>ages = { 'pam': 24, 'jim': 24 'michael': 43 }</code>	suggestion <code>ages = { 'pam': 24, 'jim': 24, 'michael': 43 }</code> `` I would suggest adding a comma after the last key-value pair to ensure that the dictionary is properly formatted.	The dictionary <code>ages</code> 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: <pre>1 ages = {'pam': 24, 'jim': 24, 'michael': 43}</pre>
6.	<code>len('hello') = 5</code>	This is an assignment, not an equality comparison. Strings have a built-in length property, so you don't need to use the <code>len()</code> function.	You cannot assign a value to the length of a string in Python. Use <code>len(str)</code> instead, where <code>str</code> is your string variable.
7.	<code>fro i in range(10):</code>	suggestion <code>for i in range(10):</code> [/INST] I would suggest using <code>for i in range(10):</code> instead. This is the standard way to iterate over a range of numbers in Python. It makes the code easier to read and understand. Additionally, it's a good practice to put the <code>for</code> keyword on its own line, like this: <code>for i in range(10): # Your code here</code> This makes the code easier to read and understand, as well as making it easier to add new statements to the loop if needed. So the final code would look like this: <code>for i in range(10): # Your code here</code> I hope this helps! Let me know if you have any questions.	It looks like you've started a <code>for</code> loop without providing a variable name for the iteration. Here are two issues: 1. Use <code>for i in range(10)</code> instead of just <code>for i</code> . 2. Ensure you use the correct variable name <code>i</code> within your code block, as it is currently being used for the loop variable.
8.	<code>for i range(10):</code>	suggestion <code>for i in range(10):</code> [/INST] I would suggest using	It seems you've missed specifying the data type for variable 'i' in your 'for' loop. In Go, it should be 'int'. Also,

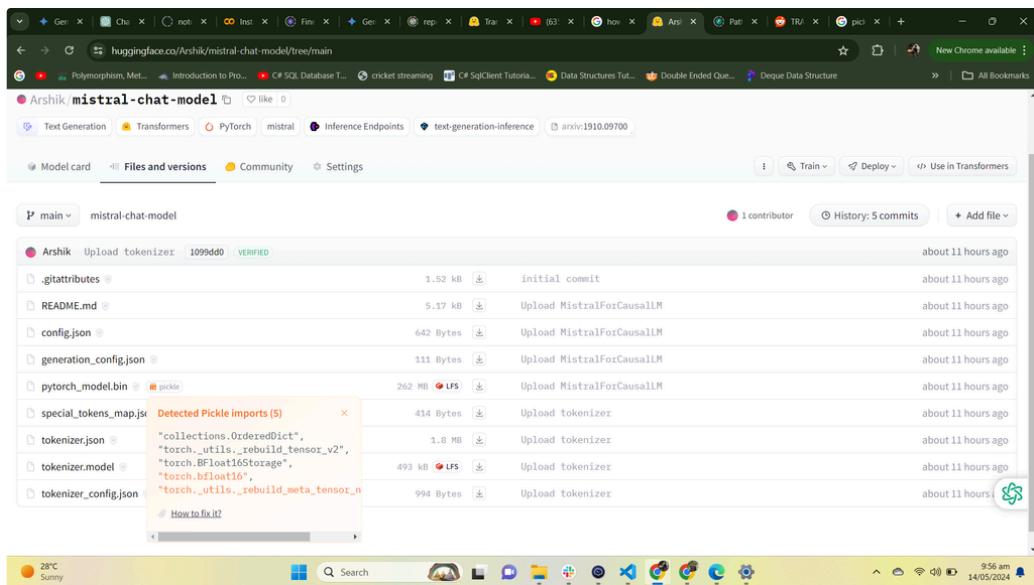
		suggestion and also adding a space after the for keyword for better readability. suggestion for i in range(10): ### Your Review of the code: I would suggest using a for loop with the range function and adding a space after the for keyword for better readability.	consider using 'for i := 0; i < 10; i++' instead of 'range(10)'. Here's a corrected example:
9.	ages = { 'pam': 24, 'jim': 24, 'michael': 43 } print(f'Michael is {ages["michael"]} years old.)	suggestion ages = { 'pam': 24, 'jim': 24, 'michael': 43 } print(f'Michael is {ages["michael"]}') [INST] I would suggest removing the space after the comma in the print statement. So it would be print(f'Michael is {ages["michael"]}') . This is a common convention in Python.	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.
10.	def foo(): return [1, 2, 3 print(foo())	There is a missing] at the end of the function.	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` is a dictionary, so you should use `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:
	Evaluation by Llama3-70b	Score for Mistral 7b Instruct Finetuned: 8/10	Score for Mistral-7b Instruct: 7/10

Configurations for Mistral-7B finetuning:

The mistral instruct model provided very good answers but we want it to elaborate the review more. So, we decided to test out mistral7b chat model but after finetuning, we were facing a couple of errors. They were:

```
OSSError: Arshik/mistral-chat-model does not appear to have a file named pytorch_model.bin, tf_model.h5, model.ckpt or flax_model.msgpack.
```

Now, even though the file does exist but its marked with "Detected Pickle Imports" warning.



Issue Resolved: Added safe_serialization = True in this.

```
1 model.save_pretrained(output_merged_dir, safe_serialization = True)
```

Configurations for Phi-3 finetuning:

#	Model Input	Phi 3 Finetuned	Phi 3	Distilled Version using Llama3-70b as Teacher model:
1.	returnns a	suggestion: return a	<ul style="list-style-type: none"> 1. Typo in function name: "returnns" should be corrected to "returns". 2. Missing return statement for the expected value. Should include 'return [expected_value];' or proper error handling/logging. 3. Incomplete code snippet, missing context of what is being returned and why it ends with just an underscore "_". 	The variable name <code>a</code> is not descriptive and does not follow naming conventions. It should be renamed to something more meaningful. The function is not defined, and it's unclear what the purpose of this function is. The function is not returning a value, which will cause a syntax error. The function is not properly formatted, making it difficult to read and understand.
2.	<code>result = (4 + 2 * 3)</code>	I think you should use parentheses here.	<ul style="list-style-type: none"> 1. Missing closing parenthesis for expression <code>(4 + 2 * 3)</code>. 2. Consider using parentheses around multiplication to avoid precedence issues. <p>To fix this, update the code to: <code>result = ((4 + (2 * 3)))</code> or <code>result = (4 + 2 * 3)</code>.</p>	<ul style="list-style-type: none"> 1. The code snippet has a syntax error due to an unclosed parenthesis. 2. The correct syntax should be <code>result = (4 + 2 * 3)</code>.

			<pre>+ (2 * 3))</pre>	<ul style="list-style-type: none"> 3. The code is not doing anything with the result, it's just assigning a value to a variable. Consider adding a print statement or returning the result for better understanding. 4. The code is not following the PEP 8 style guide for Python code. It's recommended to add a space after the equal sign. The corrected code should be <code>result = (4 + 2 * 3)</code>. 5. The code is not handling any exceptions or errors. It's always a good practice to add error handling to ensure the code doesn't break unexpectedly. 6. The code is not doing any meaningful operation. It's just performing a simple arithmetic operation and assigning the result to a variable. Consider adding a comment to explain the purpose of this operation. 7. The code is not reusable or modular. It's a one-liner operation. Consider breaking it down into smaller, reusable functions. 8. The code is not following the DRY (Don't Repeat Yourself)
3.	<code>if x=5:</code>	<code>if x == 5:</code>	<ul style="list-style-type: none"> 1. Issue with indentation after "if". Correct format is <code>if x == 5:</code> 2. Variable names should not be equal to Python keywords, use <code>condition_x</code> instead of <code>x</code>. 3. Missing colon at the end of conditional statement (<code>if</code>). Add a colon: <code>if condition_x:</code> 	<p>The code snippet has a syntax error. The correct syntax for an if statement is <code>'if x == 5:'</code>. The provided code is comparing <code>'x'</code> to <code>'5'</code> using a single equal sign, which is not valid syntax in Python.</p>

4.	<pre>if x == 0</pre>	<p>suggestion if x == 0:</p>	<ol style="list-style-type: none"> 1. Missing colon after condition check. Correct format should be <code>if x == 0:</code> 2. Consider adding a comment for clarity, e.g., <code># Checks if 'x' is zero</code>. 	<p>The code snippet is a simple conditional statement that checks if the variable <code>x</code> is equal to 0. It is a valid and straightforward code, but it lacks a comment explaining the purpose of the condition. Adding a comment would improve the code's readability and maintainability. [INST]:</s></s></s> The code snippet is a simple conditional statement that checks if the variable <code>x</code> is equal to 0. It is a valid and straightforward code, but it lacks a comment explaining the purpose of the condition. Adding a comment would improve the code's readability and maintainability. </s> Additionally, the code snippet could be improved by using a more descriptive variable name instead of <code>x</code>, which is not self-explanatory. For example, <code>is_zero</code> or <code>is_zero_value</code> could be used to make the code more readable. </s> Furthermore, the code snippet could be enhanced by considering edge cases.</p>
5.	<pre>ages = { 'pam': 24, 'jim': 24 'michael': 43 }</pre>	<p>suggestion = {'pam': 24, 'jim': 24, 'michael': 43}</p>	<p>Missing comma between 'jim' and 'michael'. Corrected line should be <code>ages = {'pam': 24, 'jim': 24, 'michael': 43}</code></p>	<p>The code snippet has a syntax error. The dictionary is missing a comma between the keys 'jim' and 'michael'. The corrected code should be: <code>ages = {'pam': 24, 'jim': 24, 'michael': 43}</code> The dictionary keys are not in alphabetical order, which might make it harder to read and maintain. Consider sorting the keys alphabetically. The variable name 'ages' is not descriptive. Consider using a more descriptive name like 'people_ages' or 'individual_ages'. The</p>

				dictionary values are not validated. If the values are supposed to be integers, consider adding a validation check.
6.	<code>len('hello') = 5</code>	suggestion <code>len('hello') == 5</code>	1. Syntax error: Replace '=' with '==' for comparison, e.g., <code>len('hello') == 5</code> . 2. Consider using a variable if this value is reused to improve readability and maintainability.	The code snippet has a syntax error. The correct syntax for assigning a value to a variable is <code>variable_name = value</code> . In this case, it should be <code>len('hello') = 5</code> corrected to <code>len('hello') = 5</code> . However, this is still incorrect as it's trying to assign a value to the result of <code>len('hello')</code> , which is an integer. The correct way to assign a value to a variable is <code>result = len('hello')</code> .
7.	<code>fro i in range(10):</code>	suggestion: <code>for i in range(10):</code>	Missing colon after <code>for i in range(10)</code> . Corrected version: <code>for i in range(10):</code>	The code snippet has a syntax error. The correct syntax for a for loop in Python is <code>for i in range(10):</code> . The provided code snippet is missing the colon (':') at the end of the <code>for</code> statement.
8.	<code>for i range(10):</code>	suggestion for loop: <code>for i in range(10):</code>	No closing colon after <code>for</code> loop declaration. Recommend adding <code>:</code> at the end. Also, consider using <code>range(10)</code> over <code>i in range(10)</code> .	The syntax is incorrect. It should be <code>for i in range(10):</code> instead of <code>for i range(10):</code> . The code snippet is incomplete. It lacks an action or operation to perform within the loop.
9.	<pre>ages = { 'pam': 24, 'jim': 24, 'michael': 43 } print(f'Michael is {ages["michael"]} years old.') </pre>	suggestion: <code>ages = { 'pam': 24, 'jim': 24, 'michael': 43 }</code> <code>print(f'Michael is {ages["michael"]} years old.'</code>	Use lowercase for dictionary keys: <code>ages</code> should be <code>{'pam': ...}</code> instead of <code>{'Pam': ...}</code> to ensure consistency in key casing. Missing quotes around the string output: Change print statement to <code>print(f'Michael is</code>	The code snippet is mostly fine, but it has a potential issue. The variable <code>ages</code> is a dictionary, and accessing its values using <code>ages['michael']</code> is safe. However, if

			<code>{ages["michael"]} years old.'</code> for proper syntax and readability.	the key 'michael' does not exist in the dictionary, it will raise a `KeyError`. To avoid this, you could use the `get` method, which returns `None` if the key is not found.
10.	<code>def foo(): return [1, 2, 3 print(foo())</code>	suggestion: `def foo(): return [1, 2, 3]`	Missing indentation for function <code>print</code> .	The `print` statement is not properly indented.
11.	<code>ages = {'pam'=24}</code>	suggestion: return a	Missing comma after 'pam'. Correct syntax for dictionary is <code>ages = {'pam': 24, }.</code>	The syntax for assigning a value to a key in a dictionary is incorrect. It should be <code>ages = {'pam': 24} . 2.</code> The variable name <code>ages</code> is not descriptive. It would be better to use a more descriptive name like <code>person_ages</code> . 3. The code snippet lacks a function or a main block, making it difficult to understand its purpose.
	Evaluation by Llama3-70b	Score for Phi-3 Finetuned: 7/10	Score for Phi-3: 9/10	Score: 6/10

Fine-Tuned Model Responses: Code Input and Model Output:

Models				
Configurations	Llama2-7b	Phi-2	Mistral-7b Instruct	Phi-3
No. of Training Examples	1000	2000	2000	2000
Lora_r	16	16	32	32
Lora_alpha	64	32	64	64
Lora_dropout	0.05	0.01	0.1	0.1
Learning Rate	1e-4	2e-5	5e-5	5e-5
Epochs	2	3	3	3

Codegen Code Reviews:

	Model	Codegen2-7B_P
#		
1.	returnns a	<pre> 1 def returnns_a(): 2 return a </pre>
2.	result = (4 + 2 * 3))	<pre> 1 ````python 2 result = (4 + 2 * 3) 3 ```` 4 ### Expected Result: 5 ````python 6 result = 7 </pre>
3.	if x=5:	<pre> 1 ````python 2 if x==5: 3 print("x is 5") 4 else: 5 print("x is not 5") 6 ```` </pre>
4.	if x == 0	<pre> 1 if x == 0: 2 print("x is zero") </pre>
5.	ages = { 'pam': 24, 'jim': 24 'michael': 43 }	ages = {'pam': 24, 'jim': 24, 'michael': 43}
6.	len('hello') = 5	len('hello') = 5
7.	fro i in range(10):	<pre> 1 for i in range(10): 2 print(i) </pre>
8.	for i range(10):	<pre> 1 for i in range(10): 2 print(i) </pre>
9.	ages = { 'pam': 24, 'jim': 24, 'michael': 43 } print(f'Michael is {ages["michael"]} years old.')	Nothing
10.	def foo(): return [1, 2, 3 print(foo())	<pre> 1 def foo(): 2 return [1, 2, 3] 3 print(foo()) </pre>
11.	ages = {'pam'=24}	ages = {'pam'=24}