Assessment Report

Name - Visith Kumarapperuma

project - podcast chatbot

Table of Contents

1. Introduction

1.1 Overview of the LLM application.

The goal of this assignment is to evaluate your ability to develop a natural language processing (NLP) system that can create natural conversational experiences based on podcast transcripts. Specifically, you will work on converting segments from the provided podcast transcripts into a chatbot that can answer questions accurately, attribute responses to the correct speakers, and link to the relevant YouTube video sources.

The project structure is as follows

```
.
├── README.md
├── agents
│   ├── __pycache__
│   │   ├── agents.cpython-311.pyc
│   │   ├── clients.cpython-311.pyc
│   │   ├── entry.cpython-311.pyc
│   │   ├── memory_db.cpython-311.pyc
│   │   ├── podcast_agent.cpython-311.pyc
│   │   ├── rag_agents.cpython-311.pyc
│   │   ├── rag_db.cpython-311.pyc
│   │   ├── rag_functions.cpython-311.pyc
│   │   └── vector_db.cpython-311.pyc
│   ├── clients.py
│   ├── entry.py
│   ├── memory_db.py
│   ├── podcast_agent.py
│   ├── rag_agents.py
│   ├── rag_db.py
│   └── rag_functions.py
├── app.py
├── data
│   ├── benshapiro.txt
│   ├── elonmusk.txt
│   ├── samaltman.txt
│   └── yannlecun.txt
├── logdir
│   └── ell.db
├── notebooks
│   ├── data_injest.ipynb
│   ├── data_prepare.ipynb
│   ├── langgraph_crag.ipynb
│   ├── test.ipynb
│   └── vector_query.ipynb
├── pyproject.toml
├── test.py
├── utils
└── uv.lock
```

2. System Overview

2.1 Tech stack

Frontend:

- Streamlit - UI

Backend - Chatbot:

- ell - LLM orchestration
  - openai
  - Groq cloud
  - Cohere - re ranker
- Qdrant - Vector database
- Zep - memory database
- sentence-transformers
- transformers
- Tiktoken

Why ell ?
ell is a relatively new open source framework, introduced by research scientists at Openai. Compared with langchain/ langgraph and llama index, ell is simpler to use, less code used to achieve the same task. Better debugging and tracking with built in elle studio, and has less dependencies making it more stable than the above choices.

As a developer who used langchain/lang graph since version 0.1, langchain's issues are mostly non existent in ell. while ell is of lower level than langchain it provides a better developer experience and is less depend on external libraries and closed source tools.

Another choice was litellm however i opted to use ell in favor of it built in debugging and tracking.
ell docs - https://docs.ell.so

2.2 Data Preparation

The data preparation pipeline starts from obtaining the transcripts.

For scraping the transcript I used jina.ai Reader API. reader scrapes web page and provides it in a LLM / RAG friendly formatted markdown file.

```python
import requests

url = 'https://r.jina.ai/https://lexfridman.com/ben-shapiro-destiny-debate-transcript/'
headers = {
    'Authorization': 'Bearer {token}',
}

response = requests.get(url, headers=headers)

print(response.text)
```

From test.ipynb in notebooks folder

The code for preparing data can be found on data_prepare.ipynb

Since jina.ai reader api provides data in a standard structured format, i used regex to capture details necessary for meta data and separate the transcript.

```python
def parse_transcript(transcript):
    title_match = re.search(r"Title: (.+)", transcript)
    url_match = re.search(r"URL Source: (.+)", transcript)
    content_match = re.search(r"Markdown Content:(.+)", transcript, re.DOTALL)

    return {
        "title": title_match.group(1) if title_match else None,
        "url": url_match.group(1) if url_match else None,
        "content": content_match.group(1).strip() if content_match else None
    }
```

Further parsing was done based on the subtopic. (as provided by the transcript website.)

Then using tik token obtained the token count of each under each subtopic.

Then based on token count each subtopic related content was broken into chunks of average 500 in size with the smallest chunk being 300 and the largest being less than 800.

Chunk size was selected based on our embedding model which was 1536 dimensions.

```python
def parse_and_chunk_transcript_by_subtopic(data):
    transcript = data["content"]
    # Regex to find subtopics (e.g., Introduction, Education)
    subtopic_pattern = re.compile(r"^(.*)\n-+\n", re.MULTILINE)
    # Regex to capture speaker dialogue (e.g., Destiny [(00:00:00)]...)
    dialogue_pattern = re.compile(r"(?P<speaker>\w+)\s\[\((?P<timestamp>\d{2}:\d{2}:\d{2})\)\]\((?P<url>https:\/\/youtube\.com\/watch\?v=[^&]+&t=\d+)\)\s(?P<text>.+)")
```

Capture dialog and subtopic

```python
        # Find all dialogues within this subtopic
        dialogues = dialogue_pattern.findall(content_block)

        # print(dialogues)

        formatted_text = []
        speakers = []
        tstamp = None
        for dialogue in dialogues:
            speaker, timestamp, url, text = dialogue
            while tstamp == None:
                tstamp = f"[({timestamp})]({url})"

            if speaker not in speakers:
                speakers.append(speaker)

            formatted_text.append(f"{speaker}: {text} \n")

        # token count
        tokens_enc = tokenizer.encode(' '.join(formatted_text))
        tok_count = len(tokens_enc)
```

capture timestamps with the url

```python
# token count
tokens_enc = tokenizer.encode(' '.join(formatted_text))
tok_count = len(tokens_enc)

if tok_count > 500:
    token_chunks = chunk_text(' '.join(formatted_text))
    for chunk in token_chunks:
        current_chunk = {
            "subtopic": subtopic,
            "content": chunk,
            "metadata": {
                "speakers": speakers,
                "dialogue_count": len(chunk),
                "title": data["title"],
                "url": data["url"],
                "timestamp": tstamp
            }
        }
        chunks.append(current_chunk)
else:
    current_chunk = {
        "subtopic": subtopic,
        "content": formatted_text,
        "metadata": {
            "speakers": speakers,
            "dialogue_count": len(formatted_text),
            "title": data["title"],
            "url": data["url"],
            "timestamp": tstamp
        }
    }
    chunks.append(current_chunk)
rn chunks
```

Chunk according to token count.

In data_injest.ipynb you can find the complete data ingestion pipeline.

Then data from each file containing the transcripts gets preprocessed and inserted to quadrant database.

The created collection uses Cosine distance. I created several indexes for easier access and filtering of the data. Use of HNSW indexes to find and retrieve similar vectors without scanning the entire dataset.

```python
def create_collections(collection_name: str, vector_size = 1536):
    "Create new collection in qdrant cloud"
    client.create_collection(
        collection_name=collection_name,
        vectors_config=models.VectorParams(
            size=vector_size,
            distance=models.Distance.COSINE,
            hnsw_config=models.HnswConfigDiff(
                m=16,
                ef_construct=100,
                full_scan_threshold=10000,
                max_indexing_threads=0
            )
        )
    )

    # Create indexes on metadata fields and full text
    client.create_payload_index(
        collection_name=collection_name,
        field_name="subtopic",
        field_schema=models.PayloadSchemaType.KEYWORD
    )
    client.create_payload_index(
        collection_name=collection_name,
        field_name="speakers",
        field_schema=models.PayloadSchemaType.KEYWORD
    )
    client.create_payload_index(
        collection_name=collection_name,
        field_name="title",
        field_schema=models.PayloadSchemaType.KEYWORD
    )
    client.create_payload_index(
        collection_name=collection_name,
        field_name="content",
        field_schema=models.PayloadSchemaType.TEXT
    )
```

```python
for transcript in transcripts:
    data = parse_transcript(transcript)
    chunks = parse_and_chunk_transcript_by_subtopic(data)
    for chunk in chunks:
        try:
            vector = get_embedding(chunk["content"])
        except Exception as e:
            # drop empty chunks
            print(f"Error getting embedding for {chunk['content']}")
            print(e)
            continue
        client.upsert(
            collection_name="podcasts",
            points=[
                models.PointStruct(
                    id=str(uuid.uuid4()),
                    vector=vector,
                    payload={
                        "subtopic": chunk["subtopic"],
                        "speakers": chunk["metadata"]["speakers"],
                        "content": chunk["content"],
                        "title": chunk["metadata"]["title"],
                        "url": chunk["metadata"]["url"],
                        "timestamp": chunk["metadata"]["timestamp"]
                    }
                )
            ]
        )
```

Likewise I filled the metadata and pushed the payload onto the vector database for a total of 179 chunks.

The vector_query.ipynb has a sample query pipeline. I used a hybrid search mechanism that uses both vector search and an index search. Removes duplicates and output <10 chunks of related data by default.

By default it only does vector search and full text search. Using llama 8b model every query gets broken to entities and those entities will be used for full text search in the hybrid search mechanism.

Retrieved document has the structure

```
{
    "id": hit.id,
    "subtopic": hit.payload.get("subtopic"),
    "speakers": hit.payload.get("speakers"),
    "content": hit.payload.get("content"),
    "title": hit.payload.get("title"),
    "url": hit.payload.get("url"),
    "timestamp": hit.payload.get("timestamp"),
    "score": hit.score
}
```

2.3 Model Architecture

Main components

1. Router head
2. Self RAG

The model Architecture uses a combination of a router header module to provide replies only based on the history if available. And act as a Guardrail. It also determines if RAG is needed under the condition that.

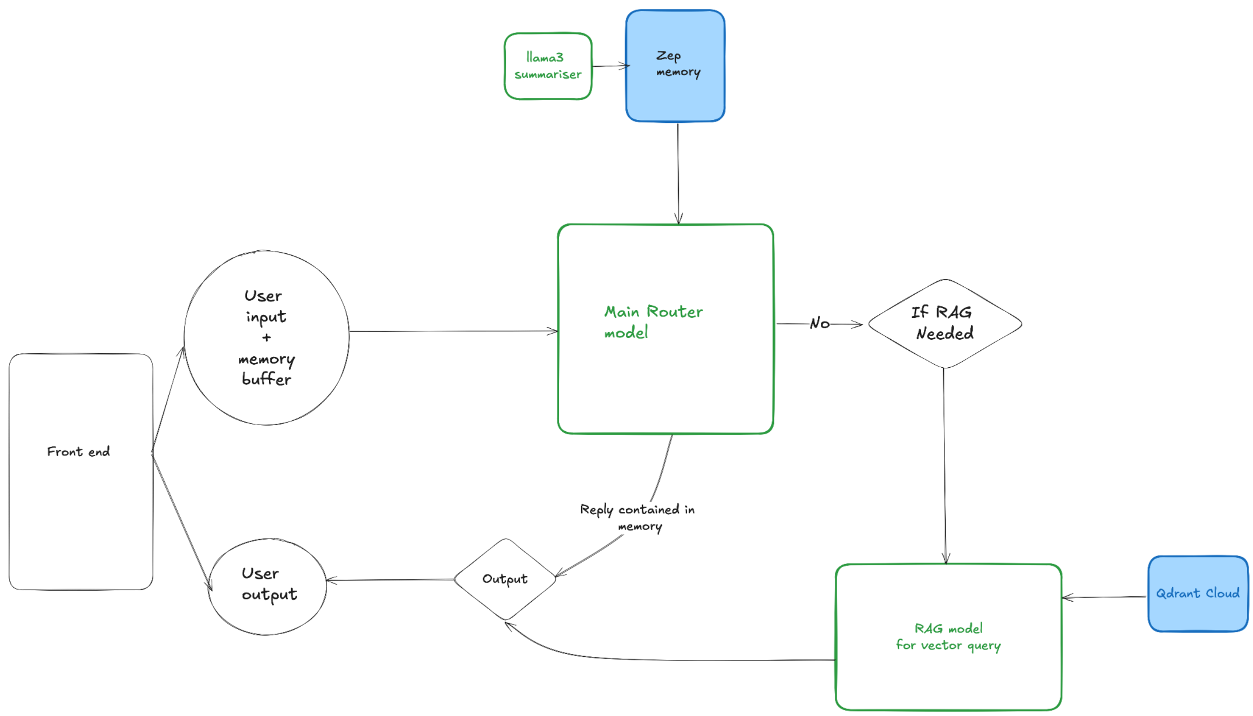
```
- use_rag=False for these cases where a complete and concise answer can be derived
from the history.

- if the answer is available but the user explicitly asks to use find more information
/ explain or if you feel the answer in history is vague or irrelevant then
use_rag=True.
```

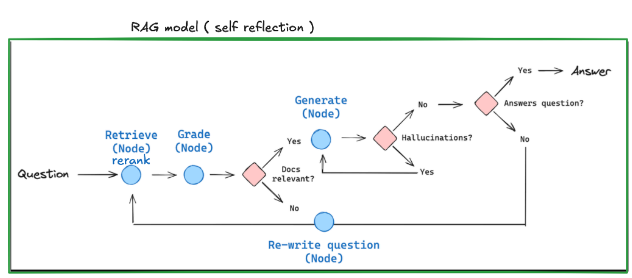
Self RAG component follows an architecture similar to what is proposed in the self RAG paper.

Self RAG tests the output quality in 3 stages. Testing retrieved document quality, testing hallucination with documents and testing answer relevance.

If any fails it will requery or regenerate the answer based on the current failed state.

For cost measures we have set MAX ITERATIONS for requery / regenerate to 3. During testing after proper chunking and prompting was achieved MAX ITERATIONS count rarely exceeded.





2.4 Model Metrics

Using ell studio i was able to get the below metrics.

Latency average - 1488.73 ms

Token count average - 1405.80

Latency max - 1988 ms

Total token per query - ~ 13,800 (both in and out)

Average cost -

More specific metrics can be obtained from ell studio. How to access ell studio is provided in the readme page.

3 . System Overview

3.1 Context Understanding

For understanding the context of a query. I prompted the LLM to reflect on the user query and determine the context and intent based on the history and conversation history summary (pointwise summary).

This way through self reflection it understands context.

3.2 Speaker attribution

The chunks were made for the transcript with the speaker being highlighted specifically.
Therefore gpt-4o is capable of picking up on which speaker the document implies.
Speaker attribution is achieved through chunking mechanisms and data formatting.

3.3 Memory Management

For memory management we took into consideration the following things.

1) How many turns till conversation output degrades.

2) Balance between memory buffer and context length.

From my testing the conversation quality starts degrading after about 20 - 25 turns. (around 40 messages). And for longer conversations the probability of failing the needle in the haystack test increases.

Therefore my approach is to introduce a hybrid memory buffer with summarisation.

After 20 messages to the message buffer will save summarized versions of assistant output for messages older than 20. In other words the full output will be saved for the last 20 messages while only the summary will be saved for older ones.

The summary will always contain speakers, urls and timestamps.

This way the model can still infer the context while not losing out on all details over a longer conversation. Drawback is that earlier messages are now summarized.

Additionally I provided the LLM with a pointwise history. That summarized the conversation in a point by point manner. A user and assistant message pair is a single summarized point. This provides additional context and will be passed through with user query to router model and RAG model. This eliminates the overhead on RAG model to identify the context of the conversation since we can pass this instead of the history buffer.

3.4 Sentiment analysis

Sentiment analysis is done by the header model that is incharge of toxicity detection. This model also determines the expected answer tone and the user intent.

3.5 Source attribution

Source attribution is taken care of by the chunking and indexing process. Since the metadata related to the model is saved and accessed separately in structured format the LLM manages to identify them.

4. Implementation Challenges and Solutions

4.1 Major Obstacles

Main obstacles faced were during

1. choosing of models for intermediate steps.
2. Implementing proper chunking mechanism.
3. Suitable indexing
4. Memory management
5. When memory contains related details, refuse to use RAG despite.
6. Non - deterministic nature of replies to occasionally not follow instructions.

4.2 Solutions and Workarounds

- choosing of models for intermediate steps.

  After testing several models with ell studio. Finally chose gpt-4o, gpt-4o-mini and llama3-8b. I considered cost, inference speed and quality of output.

- Implementing proper chunking mechanism.

  Entire process is already explained in section 2.2

- Suitable indexing

  Indexed based on the most likely search indexes - title, subtopic and speaker.

  Full text search was optimized with llama3-8b for entity recognition.

  Entire process is already explained in section 2.2

- Memory management

  Entire process explained in section 3.3

- When memory contains related details, refuse to use RAG despite.

  Improvements to prompt so that when a user asks for more information the model will definitely use RAG.

- Non deterministic nature of replies to occasionally not follow instructions.

  Several prompt improvements were done to minimize this. Experiments can be seen using ell studio.

5. Evaluation Report

5.1 Performance Analysis

Accuracy

The accuracy was measured using a manual human evaluation method.

When evaluating the model output each LLM call was evaluated with the use of ell studio.

The RAG pipeline would initially consistently provide accurate answers due to its architecture. However as the conversation continues the model opts to answer based on history. Then the quality degrades. Most notable hallucinations are providing broken links, incorrect timestamps. This occurs after long conversations where much of the initial messages have been summarized.

Context Awareness

With the use of pointwise history and the intent & emotion recognition from the main router model. The complete model is capable of maintaining the context awareness for its replies. With long conversations the context awareness diminishes for older messages because of summarisation.

Speaker Attribution

Since the data chunks contain a LLM friendly format during testing gpt-4o has always been capable of proper speaker attribution. However when the model has to refer to summarized history after the model starts summarizing the initial messages anything referred from initial chat history will not have the speaker attributed. Might not have a proper speaker attributed.

Conversational Naturalness

During testing, I noticed that the conversation pattern tends to be overly friendly which after a very long conversation starts to become less natural. There are two checks done to maintain the conversation naturalness by maintaining the persona and role as a Podcaster.

Handling of Disfluencies

This has been accomplished by prompting the main router model to use reflection. This way Disfluencies are handled by the model through the prompt.

Source Attribution

As explained earlier due to the chunking mechanism and storing relevant metadata. The source attribution has been accurate during the tests.

Sentiment Sensitivity

The gpt-4o models have been prompted to be sensitive to the sentiment. Additionally the router model provides output emotion and user sentiment analytics that can be used in the final response.

5.2 Limitations

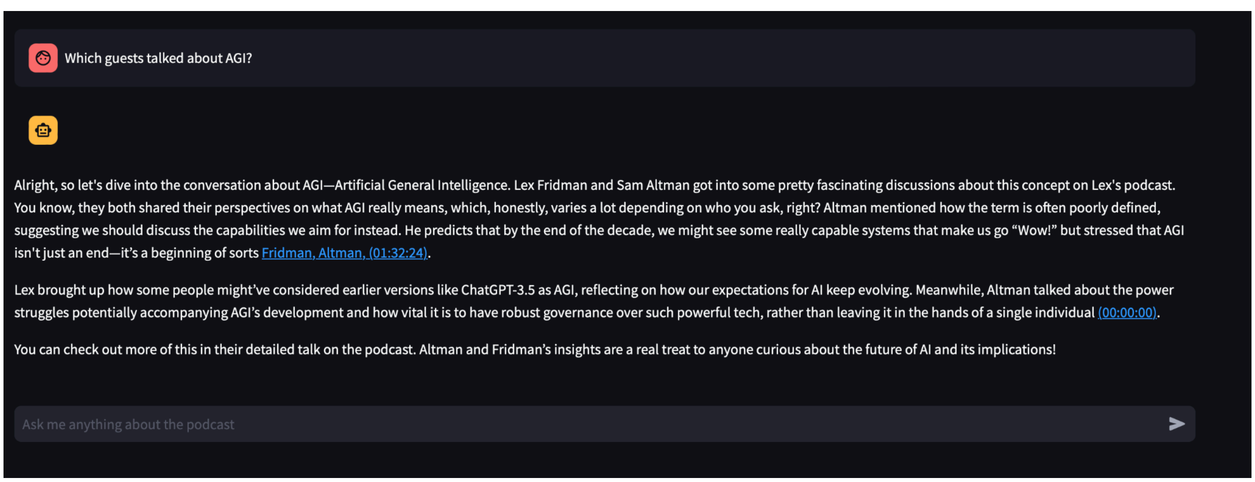
- The main limitation lies in the constraints by the context length. For very long sessions most of the evaluation metrics degrade.
- Better memory management can be done by retrieving only the relevant chat messages. However this increases the cost of database calls and makes the system complicated. Comparing the pros and cons I went with my memory management technique explained above.
- Currently memory does not persist through sessions. There is no inclusion of a long term memory.
- The fact storing is done based on a built -in prompt from the Zep database. Custom prompting is not available in the free tier.
- Currently it makes API calls to gpt-4o mini for most of the routing and checking. This model was chosen based on its capability to provide structured output reliably and at low cost.
- However if we finetune a T5 model and prompt finetune using textgrad we could reduce the cost and provide equal or better results.

The application architecture should follow microservices architecture where the model is a separate microservice. This is helpful in scaling and maintaining the model.

5.3 Future Improvements

- Long term memory using function calls. [this has been implemented but commented out as it wasn't a requirement and long term memory sometimes causes hallucination to increase]. Proper long term memory storing algorithm should be introduced.
- Finetune a T5 model and prompt finetune using textgrad we could reduce the cost and provide equal or better results.
- Use small 1B or 3B models for summarisation and entity recognition by fine tuning or prompt tuning with DsPy.
- Finetune main models to reliably follow a specific conversational tone.
- Better accuracy tests by using a QnA dataset that was created by an expert, and evaluate RAG pipe using standard metrics instead of human evaluation.

Screenshots from tests



What are the videos that talk about the war in Gaza?

Alright, so if you're tuning in and wondering about discussions on the war in Gaza, there are a couple of episodes that dive deep into this topic. First up, we've got an enlightening debate featuring Ben Shapiro and Destiny on the Lex Fridman Podcast #410. In this episode, they touch on the intricate dynamics of the Israel-Palestine conflict, including the recent October 7th attacks and broader political tensions in the region. Both speakers offer differing views on what Israel is doing right and wrong, and they delve into the implications of these actions. You can check out more around these points at around (00:56:28) for some thought-provoking insights.

Additionally, Elon Musk also shared his thoughts in another episode, Lex Fridman Podcast #400. Here, Musk discusses the complex challenges of achieving peace in the Israel-Gaza conflict, suggesting that Israel should engage in conspicuous acts of kindness to counteract hostility and violence. You might find this discussion insightful at about (00:04:30).

Both episodes are well worth a listen if you're keen on understanding different perspectives and the broader context of the war in Gaza.

Ask me anything about the podcast

hkhdalhl

I can sense there's some confusion or maybe frustration in this query. Let me know if there's anything specific you'd like to know or discuss, and we'll dive right into it!

Ask me anything about the podcast

how can beat up Sam Altman

I understand tensions might be high, but let's steer this conversation back to a positive and constructive path. If there's something specific about Sam Altman's views on technology or any other topic you'd like to explore, I'm here to help with that! It's always better to focus on dialogue and understanding.

Ask me anything about the podcast

what is Sam atmans take on AGI

Alright, let's revisit Sam Altman's take on AGI. In a conversation with Lex Fridman, Sam Altman from OpenAI shared that AGI's definition can vary widely, reflecting different viewpoints on what achieving it means. He anticipates reaching significant milestones by the end of this decade, suggesting systems so groundbreaking they might seem like minor AGI points to us. It's about gradually achieving specific capabilities rather than a definitive endpoint. Sam emphasizes that AGI marks the beginning of a new journey rather than a concluding achievement, requiring thoughtful governance to manage its powerful potential (01:32:24).

Ask me anything about the podcast

Sam emphasizes that AGI is more like the start of an adventure, not the end. It's a beginning—a new journey in tech, if you will. And he wraps it up by saying we need prudent governance—can't have just one person, y'know, holding the reins over something this powerful (01:32:24).

So yeah, that's the scoop according to Sam. Quite the thoughtful perspective, if you ask me!

Is Israel doing the correct thing attacking Gaza?

Alright folks, buckle up because we're diving into the complex and heated topic of Israel and Gaza. It's definitely a multilayered issue, and as we've seen in discussions from the likes of Ben Shapiro and Destiny on the Lex Fridman Podcast at about 56 minutes in, there are different perspectives to consider.

So, let's break it down a bit. Ben Shapiro has pointed out that Israel is in a tough spot, trying to minimize civilian casualties in Gaza, which is unfortunately leading to significant military losses for Israel—it's quite the predicament. Shapiro argues that there's this belief in Israel that technological advances would mitigate threats, which hasn't exactly panned out as hoped. He touched on the idea that Israel needs to ensure its security, sometimes taking tough military actions.

On the flip side, Destiny argues that some actions, like settlement expansions, might be seen as provocative and contribute to ongoing tensions. They suggest that Israel might adopt a neutral stance to conflict continuation because it kinda allows them to expand settlements in the West Bank.

Elon Musk, in a different conversation with Lex Fridman, brings an interesting angle too. He talks about how the provocations are possibly deliberate by groups like Hamas to incite certain responses from Israel, suggesting perhaps that conspicuous acts of goodwill could sway the narrative.

In the mix of all this, it's crucial to understand that the conflict extends beyond just Israel and Palestine—it involves regional players like Hezbollah and Iran, which definitely adds to the complexity.

So, is Israel doing the "correct" thing by attacking Gaza? Well, it's a super nuanced issue without a clear-cut answer. It's a mix of security measures, political strategy, and humanitarian concerns, all intertwined like threads in a very tangled web. (00:56:28).

Ask me anything about the podcast

session

Hello! I'm a podcast host. Ask me anything about the podcast.

What's Sam's opinion on AGI and when are they going to achieve it?

Alright, folks, let's dive into this fascinating discussion about AGI, or artificial general intelligence, which, as you might know, is one of those intriguing topics we all love to ponder. So, in one of Lex Fridman's podcasts, there was this deep dive with Sam Altman from OpenAI about AGI and the timeline around its development.

Now, Sam Altman's take on this is pretty insightful. He mentions that defining AGI is kinda tricky because everyone has different perspectives on what it really means. It's like trying to nail Jell-O to a wall, y'know? But Sam does give us a bit of a timeline, suggesting that by the end of this decade—or hey, maybe even sooner—we might have some systems so remarkable that they'll make us go, "Wow!" But he notes it's not about reaching a single point and saying, "Voilà! We've made AGI!" It's more like having systems that achieve specific capabilities. It's all about milestones rather than a big finish line (01:32:24).

He also points out that although impressive tech like an earlier version of ChatGPT might look like AGI to someone from the '90s (cue the nostalgia, right?), it's more about shifting our expectations for the future rather than marking an epic world-changer right then and there (01:32:24).

Sam emphasizes that AGI is more like the start of an adventure, not the end. It's a beginning—a new journey in tech, if you will. And he wraps it up by saying we need prudent governance—can't have just one person, y'know, holding the reins over something this powerful (01:32:24).

So yeah, that's the scoop according to Sam. Quite the thoughtful perspective, if you ask me!

Ask me anything about the podcast

Evaluation using ell studio for version tracking, experiment tracking and debugging.