Building Language Model Applications

why you should still use structured outputs

Ivan Leo (@ivanleomk on twitter)

Some Questions

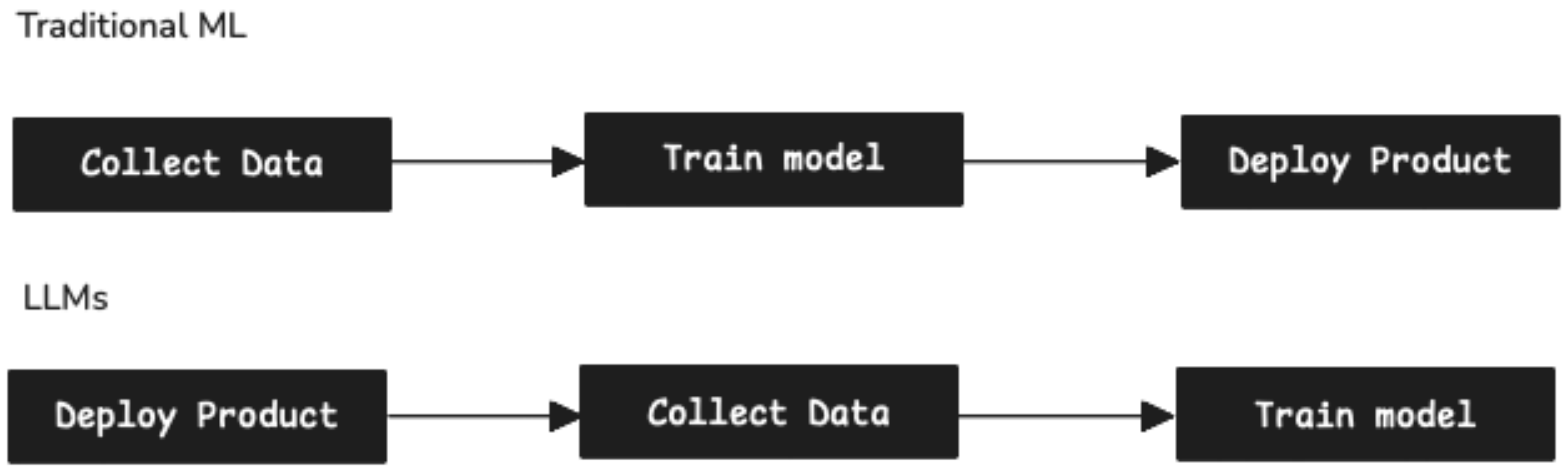
- What's so different about LLM Applications?
- How can we intelligently focus on building out application logic instead of just error handling?
- How do we know what queries our application is good at handling
- How do we systematically find weak points in our system before deployment?

Most of this content comes from a post I wrote called Taming Your LLM Application

How LLMs Flip The Script

- Instead of waiting months and spending 100-200k+ USD for a dataset, you can deploy early

- Synthetic data allows us to generate test cases before we go to production. This continues even after we've deployed.



Traditional ML

Collect Data → Train model → Deploy Product

LLMs

Deploy Product → Collect Data → Train model

Why care about Function Calling?

- It eliminates a whole new class of errors such as inconsistent outputs, parsing errors and more

- It's not the same as JSON mode - which tries to just get the model to output valid JSON

Welcome to the OpenAI Developer Forum!

What to know before posting a new question:

1. Search the forum for similar topics - the question might have been discussed before.
2. If the question relates account issues (e.g., billing and login issues), please contact us through our Help Center.
3. Please be kind and helpful in conversations!

Json format causes infinite "\n \n \n \n" in response

API gpt-4, api, json-mode

ashleyhuang707 Nov 2023

I am trying to use json format to get json response, it worked well when I give same short example, but when I use production data to test it, prompt_token = 2966, then it start to response with all "\n \n \n \n", till max token. Then, I use the same prompt without response_format, it worked well though it's not a json object.

(test with a another shorter user message still failed, is that the "#" matter?)

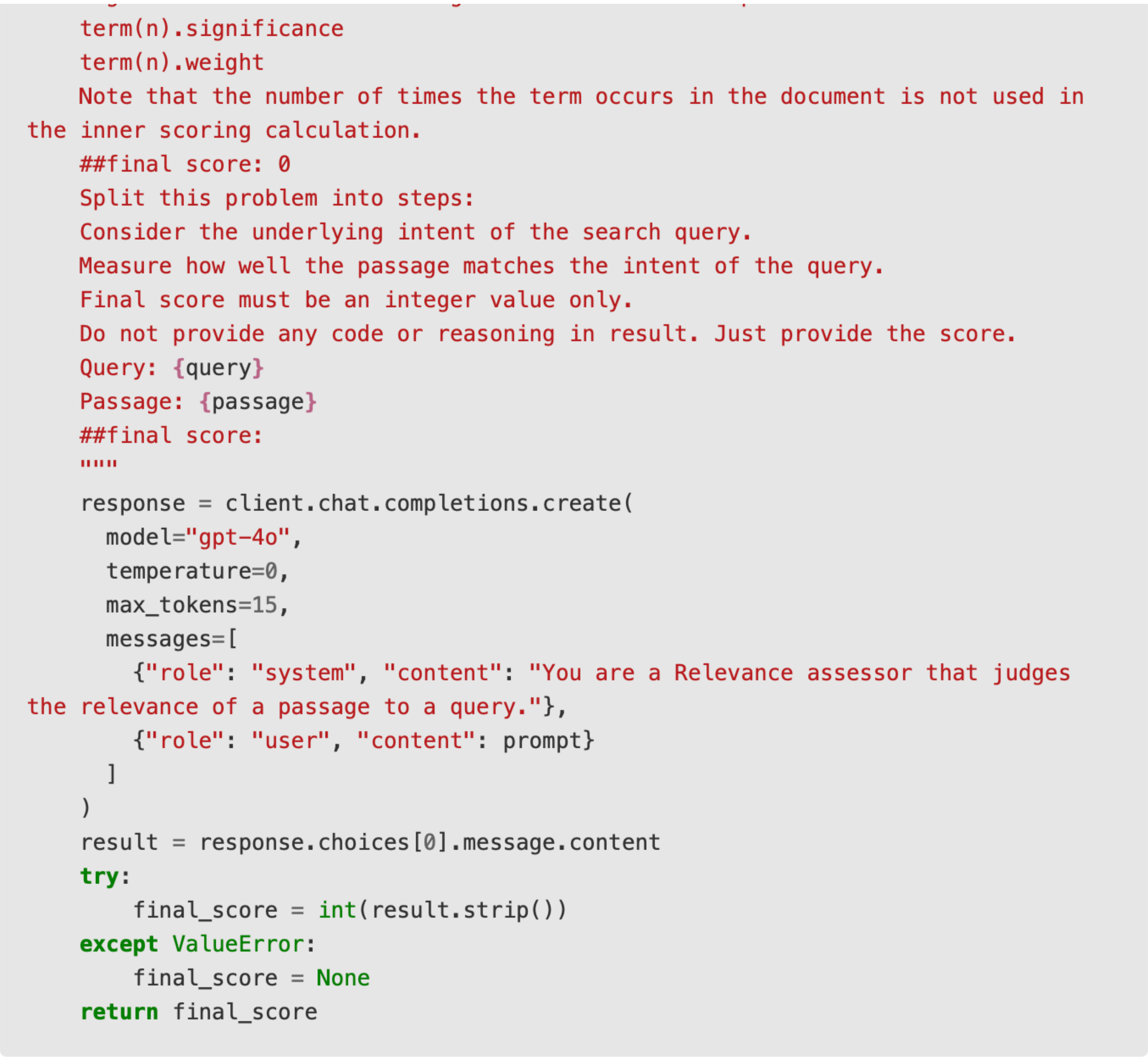
model: gpt-1106
usingAzure: True

To Reproduce

```
    here is a list of puppy selling websites: www.happ
Here is an example of the output json object:
[{
    "task_title": "1. know what puppy you want",
```

Manual JSON parsing is Hard

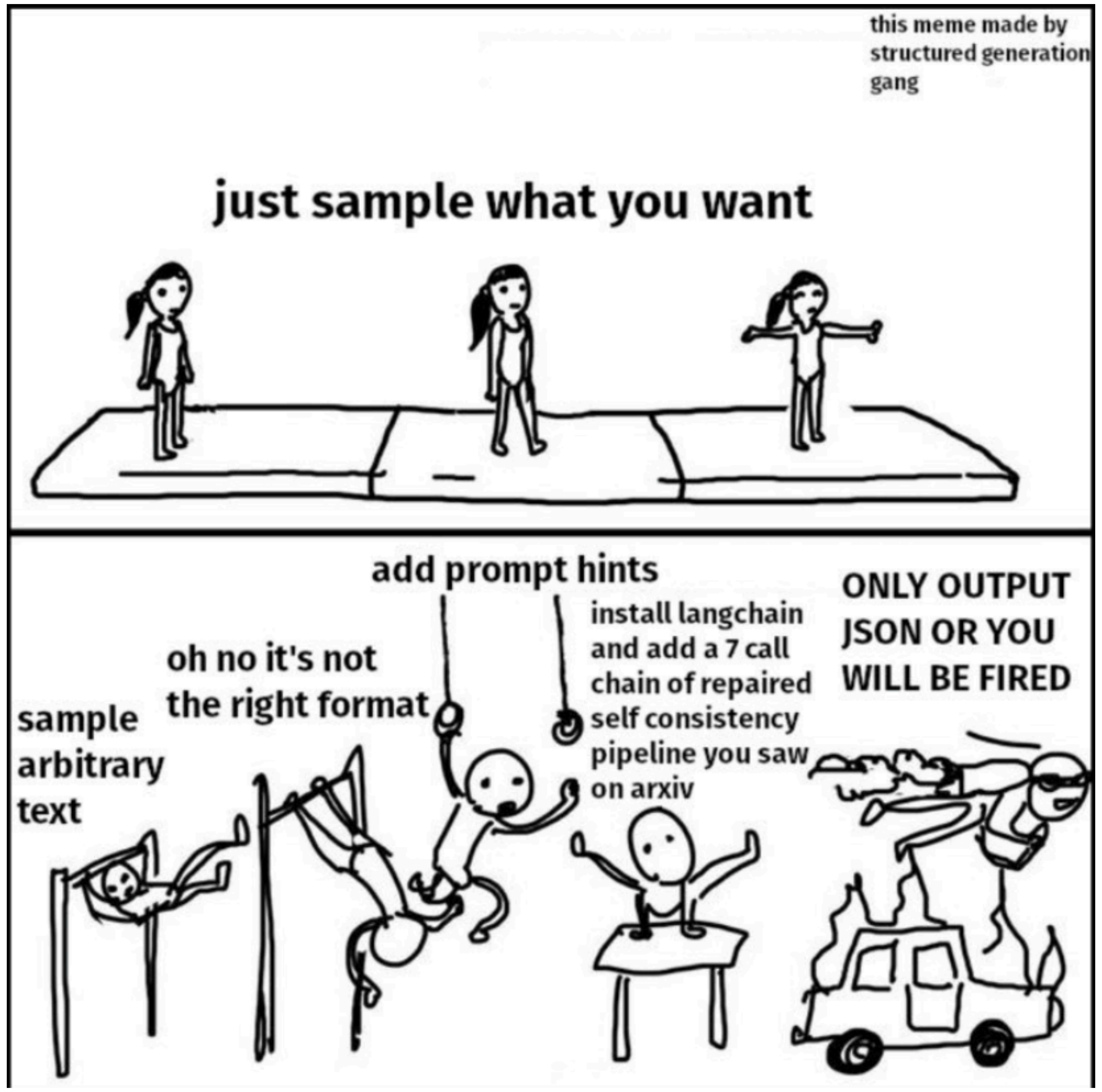
- Manually parsing the result of the JSON output is a brittle way to get the output you want
- What happens here if the language model outputs something like **AS A LANGUAGE MODEL I THINK THE ANSWER IS GOING TO BE -23**
- Also difficult to iterate and experiment with different response formats



```
    term(n).significance
    term(n).weight
    Note that the number of times the term occurs in the document is not used in
the inner scoring calculation.
    ##final score: 0
    Split this problem into steps:
    Consider the underlying intent of the search query.
    Measure how well the passage matches the intent of the query.
    Final score must be an integer value only.
    Do not provide any code or reasoning in result. Just provide the score.
    Query: {query}
    Passage: {passage}
    ##final score:
    """
    response = client.chat.completions.create(
      model="gpt-4o",
      temperature=0,
      max_tokens=15,
      messages=[
        {"role": "system", "content": "You are a Relevance assessor that judges
the relevance of a passage to a query."},
        {"role": "user", "content": prompt}
      ]
    )
    result = response.choices[0].message.content
    try:
        final_score = int(result.strip())
    except ValueError:
        final_score = None
    return final_score
```

Function Calling?

- Most inference providers support a variation of function calling (Eg. OpenAI, Gemini and Anthropic)
- You can get up to 60% performance gains by renaming a single field
- With a model fine-tuned for that specific capability, we can get consistent outputs that make it easy to compare models



Instructor

- We do one thing well - getting you validated outputs from a model
- We provide a consistent OpenAI API that allows you to basically change between different providers with minimal code changes
- We're fully open source so if you're interested do check out our repo

```python
import instructor
from pydantic import BaseModel
from openai import OpenAI


# Define your desired output structure
class UserInfo(BaseModel):
    name: str
    age: int


# Patch the OpenAI client
client = instructor.from_openai(OpenAI())

# Extract structured data from natural language
user_info = client.chat.completions.create(
    model="gpt-4o-mini",
    response_model=UserInfo,
    messages=[{"role": "user", "content": "John Doe is 30 years old."}],
)

print(user_info.name)
#> John Doe
print(user_info.age)
#> 30
```

What can you do with Structured Outputs?

It's more than just a python class

- Since Pydantic Objects are really just vanilla Python classes, we can define methods on them
- This opens up a lot of interesting possibilities that you can work with (Eg. Running an SQL call here with it)

```
class SearchIssues(BaseModel):
    """
    Use this when the user wants to get original issue information from the database
    """

    query: Optional[str]
    repo: str = Field(
        description="the repo to search for issues in, should be in the format of
'owner/repo'"
    )

    async def execute(self, conn: Connection, limit: int):
        if self.query:
            embedding = (
                OpenAI()
                .embeddings.create(input=self.query, model="text-embedding-3-small")
                .data[0]
                .embedding
            )
            args = [self.repo, limit, embedding]
        else:
            args = [self.repo, limit]
            embedding = None

        sql_query = Template(
            """
            SELECT *
            FROM {{ table_name }}
            WHERE repo_name = $1
            {%- if embedding is not none %}
            ORDER BY embedding <=> $3
            {%- endif %}
            LIMIT $2
            """
        ).render(table_name="github_issues", embedding=embedding)

        return await conn.fetch(sql_query, *args)
```

What can you do with Structured Outputs?

It's more than just a python class

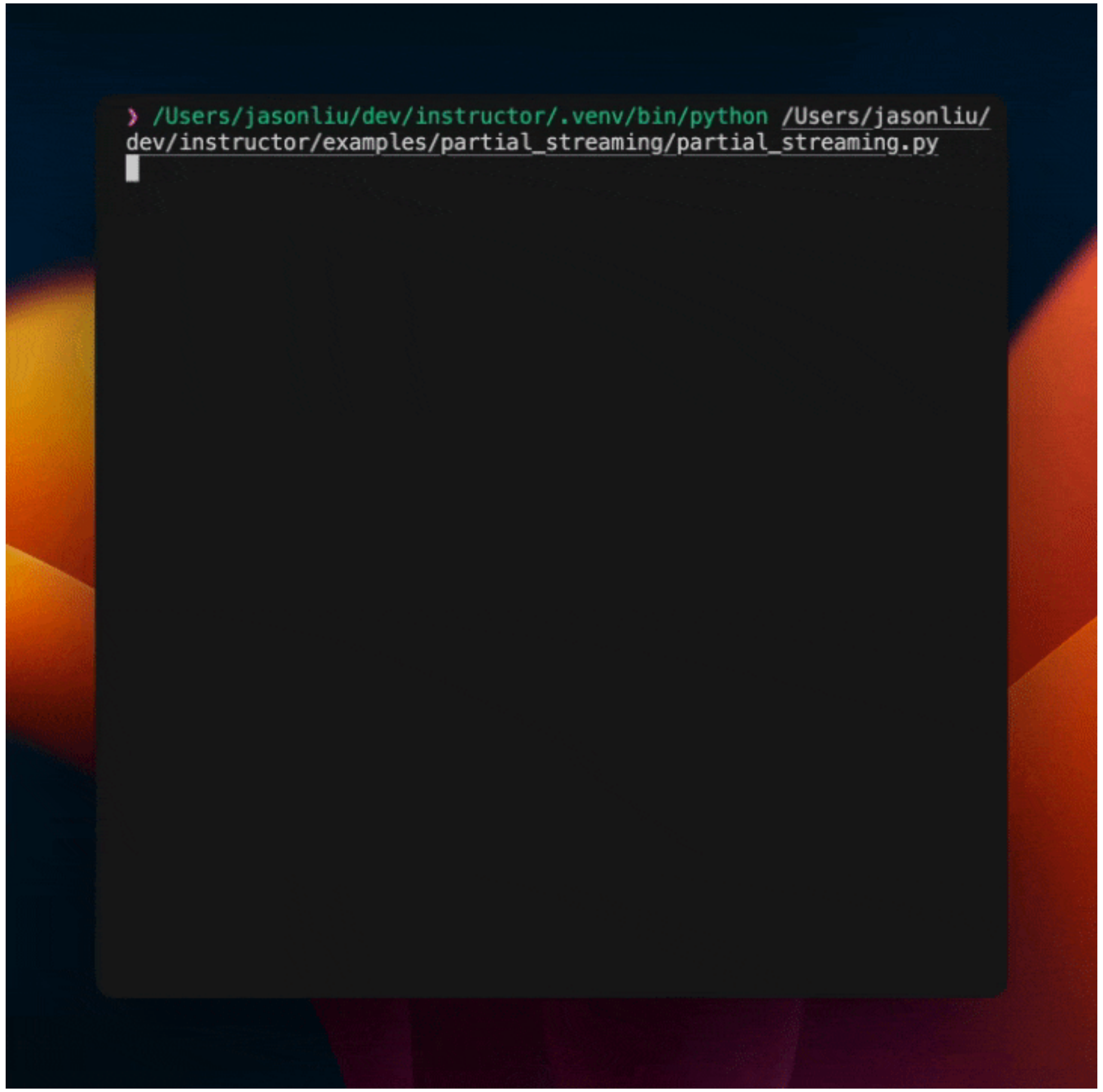
- You can also implement repair operations using libraries such as `fuzzywuzzy` to find the closest match

```
class SearchIssues(BaseModel):
    """
    Use this when the user wants to get original issue information from the database
    """

    query: Optional[str]
    repo: str

    @field_validator("repo")
    def validate_repo(cls, v: str, info: ValidationInfo):
        matched_repo = find_closest_repo(v, info.context["repos"])
        if matched_repo is None:
            raise ValueError(
                f"Unable to match repo {v} to a list of known repos of
{info.context['repos']}"
            )
        return matched_repo
```

What can you do with Structured Outputs?

It's more than just a python class

- On a downstream level, we can render and stream in the data as it comes in (with validation on a field level)
- This allows us much more safety and confidence in building out more complex features

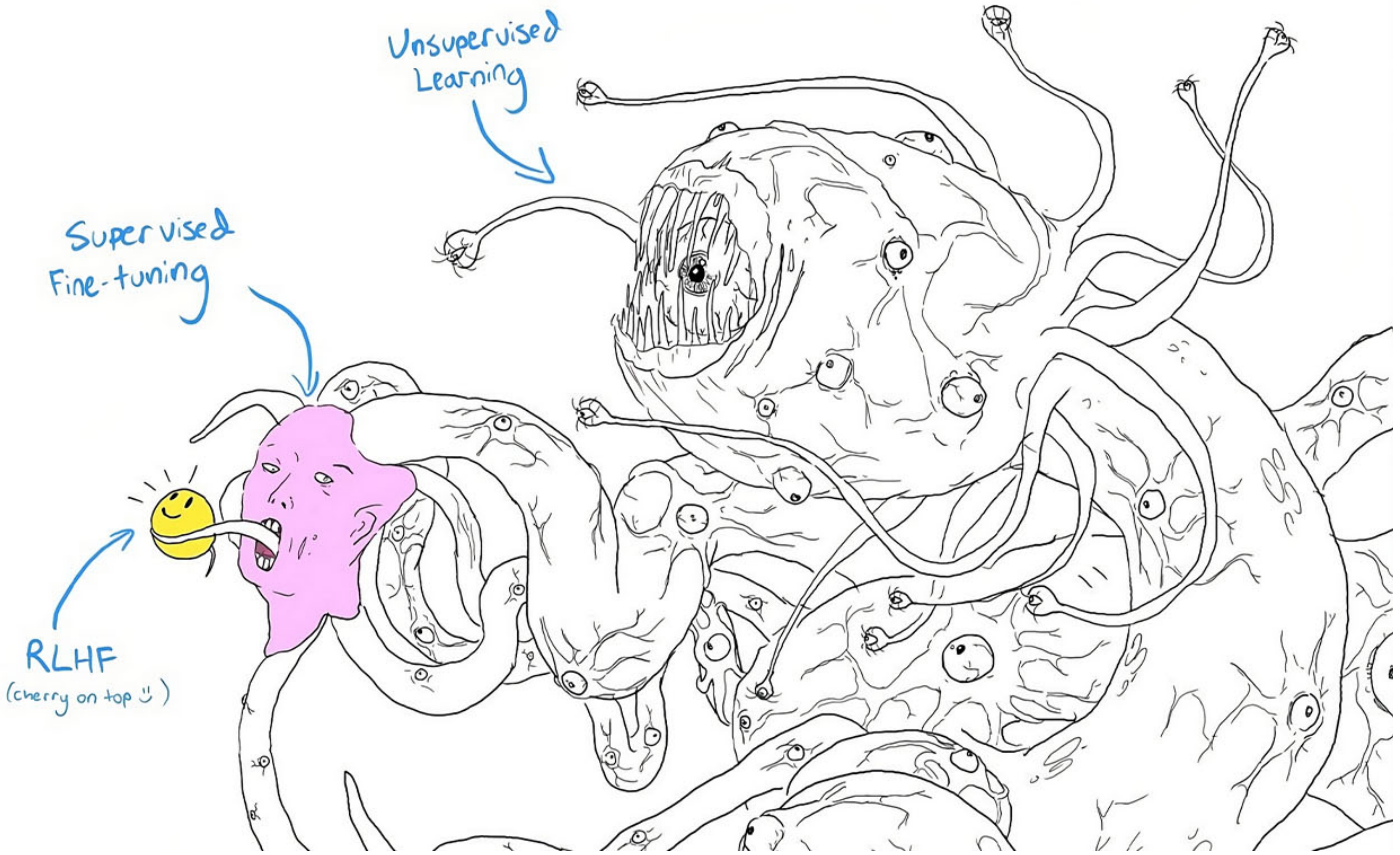


```
❯ /Users/jasonliu/dev/instructor/.venv/bin/python /Users/jasonliu/
dev/instructor/examples/partial_streaming/partial_streaming.py
```

Now that we have reliable outputs, how can we measure improvement?

Binary Metrics

- Most people like LLM as a judge to measure their RAG output, run an expensive harness to test their generated SQL code or use a human in the loop to check generated LLM response
- Starting with binary metrics gives us a way to iterate fast before we start looking at more expensive evaluation methods
- Let's look at a few examples - RAG and Text-2-SQL



RAG

- Instead of using a manual LLM as a judge to evaluate generated responses, we can measure retrieval instead

- Build towards longer context and more intelligent models instead of investing in complex infrastructure to work with today's models

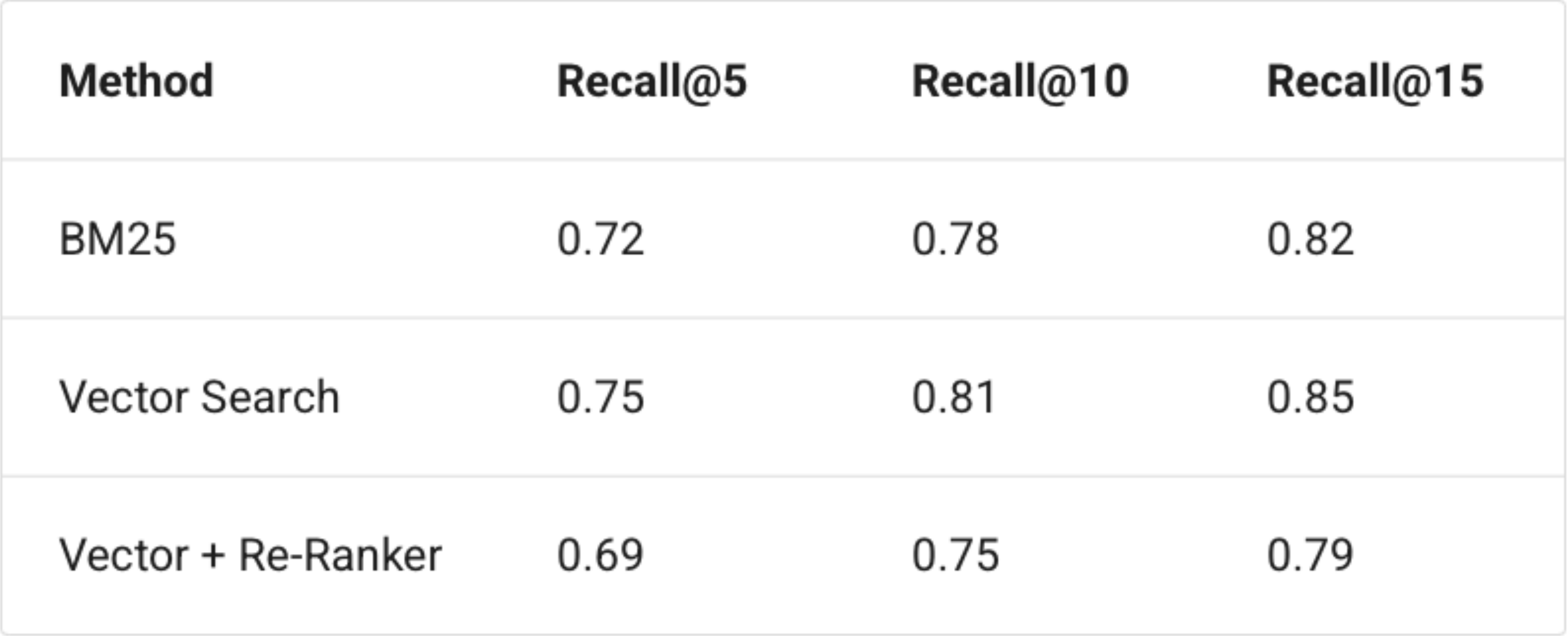
Measuring Retrieval

- We can use LLMs to generate synthetic queries for chunk(s)
- Vary them systematically using user personas, length, tone and other domain specific methods
- Test Retrieval using recall and mrr to see how different methods perform

```
import instructor
from openai import OpenAI
from pydantic import BaseModel
from rich import print

client = instructor.from_openai(OpenAI())


class Response(BaseModel):
    chain_of_thought: str
    question: str
    answer: str


chunk = """
Taiwan is a lovely island which has Chinese as its official language. Their capital
is
Taipei and the most popular dishes here are beef noodle soup, bubble tea and fried
chicken cutlets.
"""

resp = client.chat.completions.create(
    model="gpt-4o-mini",
    response_model=Response,
    messages=[
        {
            "role": "user",
            "content": (
                "Given the following chunk, generate a question that a user might
ask "
                f"which this chunk is uniquely suited to answer? \n\n {chunk}"
            ),
        }
    ],
)

print(resp)
```

Measuring Retrieval

- Let's say we run this experiment on n=100 queries and we get these results
- Then we might say that ok it seems like maybe BM25 or Full text search is a great alternative to get started
- Recall@15 for FTS is better than Recall@10 for vector search - easier/cheaper to deploy let's go ahead
- Less vibes more objective numbers



Method	**Recall@5**	**Recall@10**	**Recall@15**
BM25	0.72	0.78	0.82
Vector Search	0.75	0.81	0.85
Vector + Re-Ranker	0.69	0.75	0.79

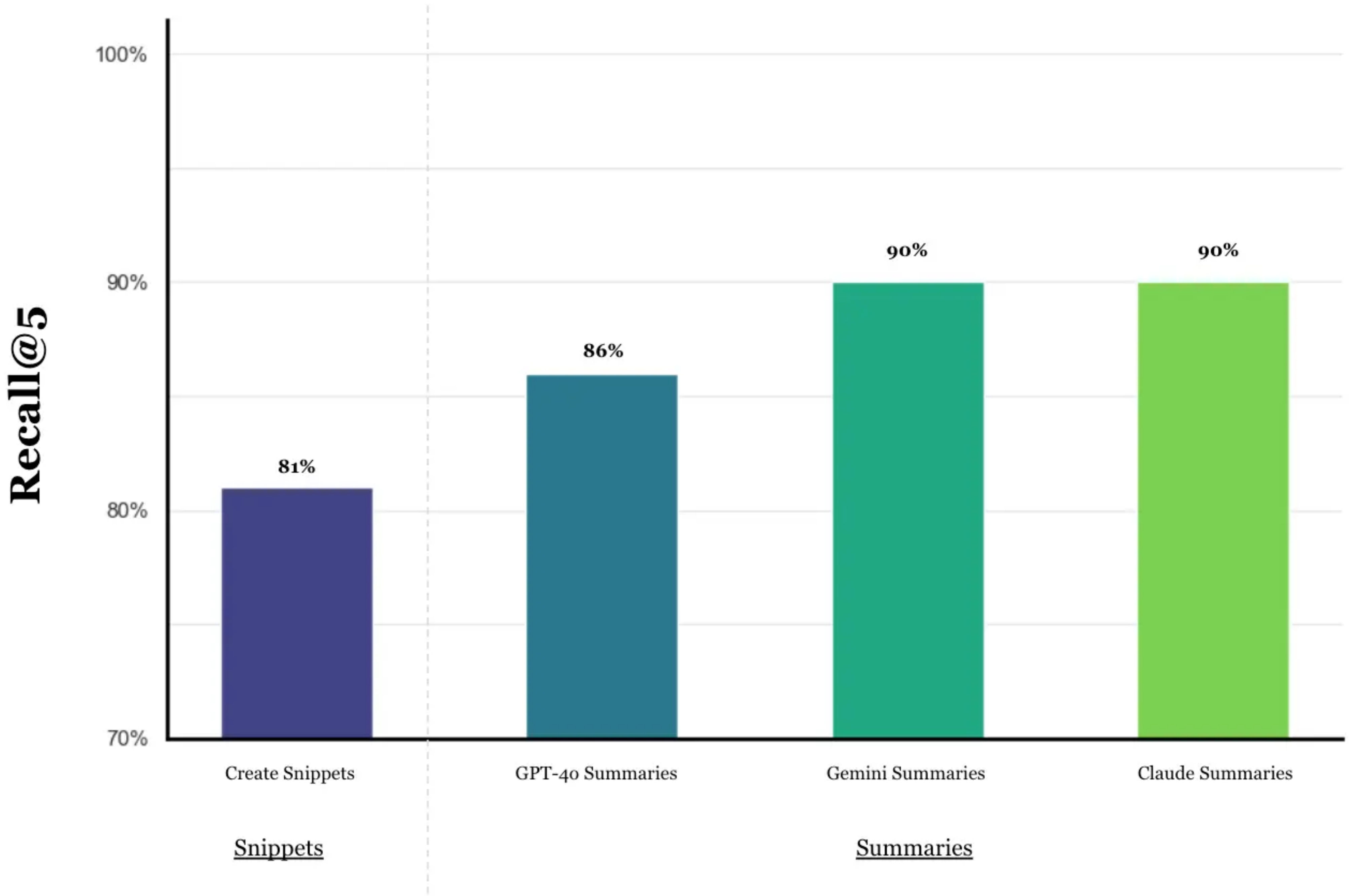
Text-2-SQL

- Generate synthetic queries for each SQL query and manually vet to make sure we're referencing at least 2-3 tables or using complex functions (Minimum complexity)
- Large Open Source datasets like BirdSQL are a good starting point
- Write questions that rely on implicit context or relationships between tables

```
SELECT
    (CAST(SUM(T1.id) AS REAL) / COUNT(T1.id)) / 4,
    T2.language
FROM
    sets AS T1
INNER JOIN
    set_translations AS T2 ON T1.id = T2.id
WHERE
    T1.releaseDate BETWEEN '2012-01-01' AND '2015-12-31'
GROUP BY
    T1.releaseDate
ORDER BY
    COUNT(T2.language) DESC
LIMIT 1
```

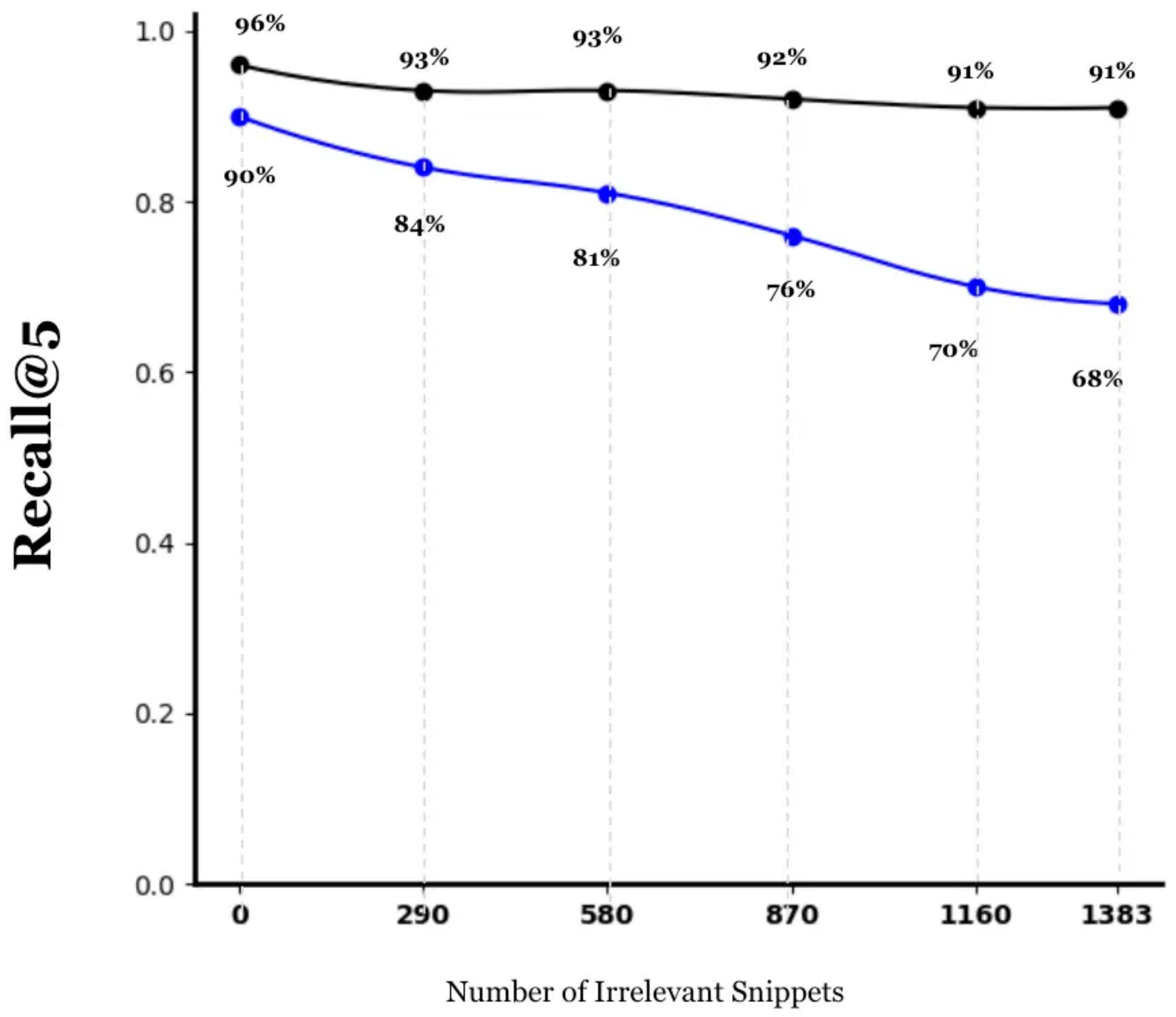
Text-2-SQL

- By doing so, we're able to take different combinations of search methods
- Evaluate all of these with precision and recall metrics too
- Validate your ideas with objective values rather than vibes~



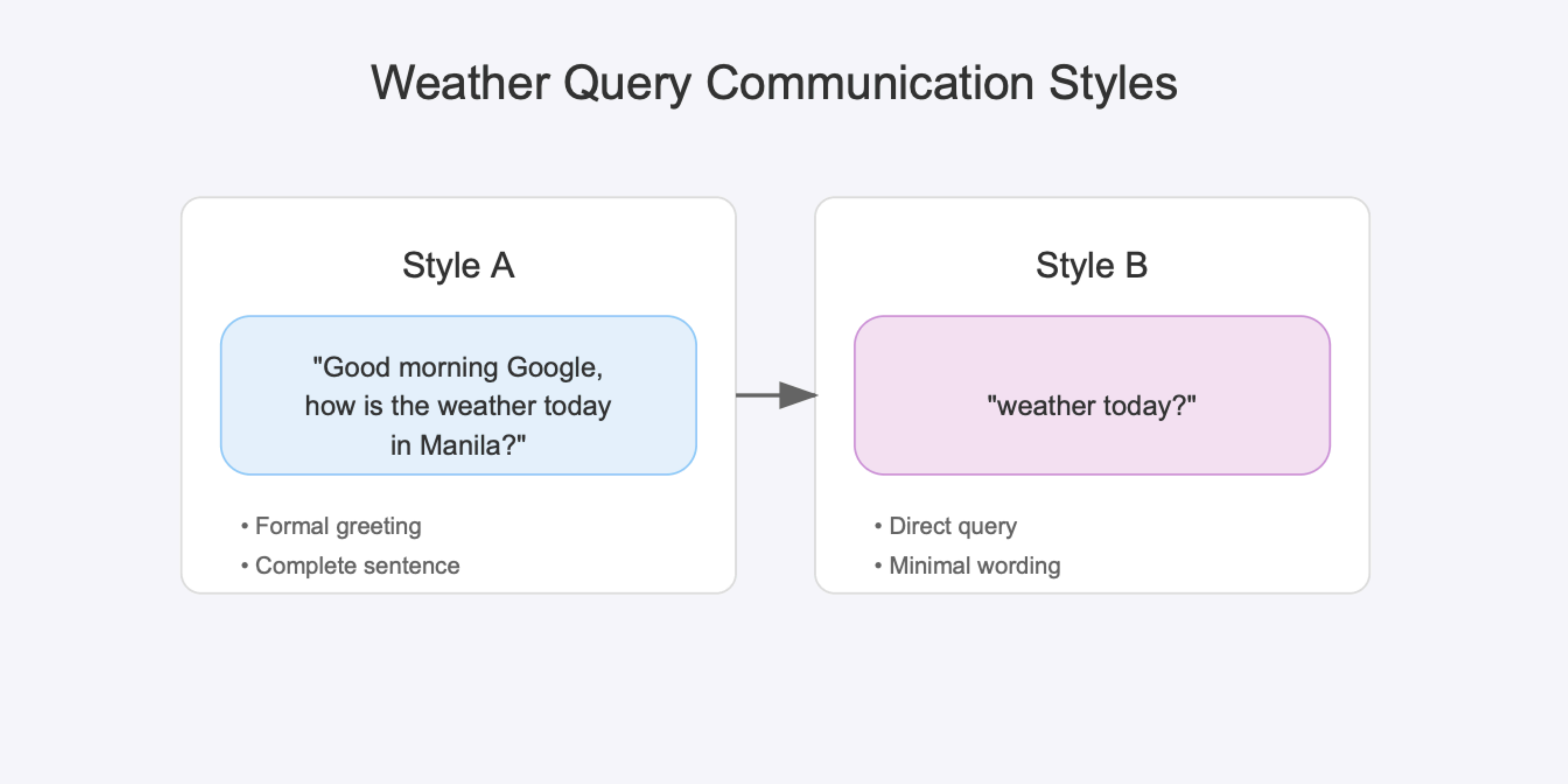
Text-2-SQL

- Without binary metrics, we might say that ok summaries are better
- But with binary metrics we can quantify the impact on recall as number of snippets increases
- We can iteratively discover the best summarisation prompt to extract inter-table relationships



Synthetic Data

- With language models, we can generate synthetic data before we even get to production
- This allows us to fuzz out how our model is going to perform at each stage



Weather Query Communication Styles

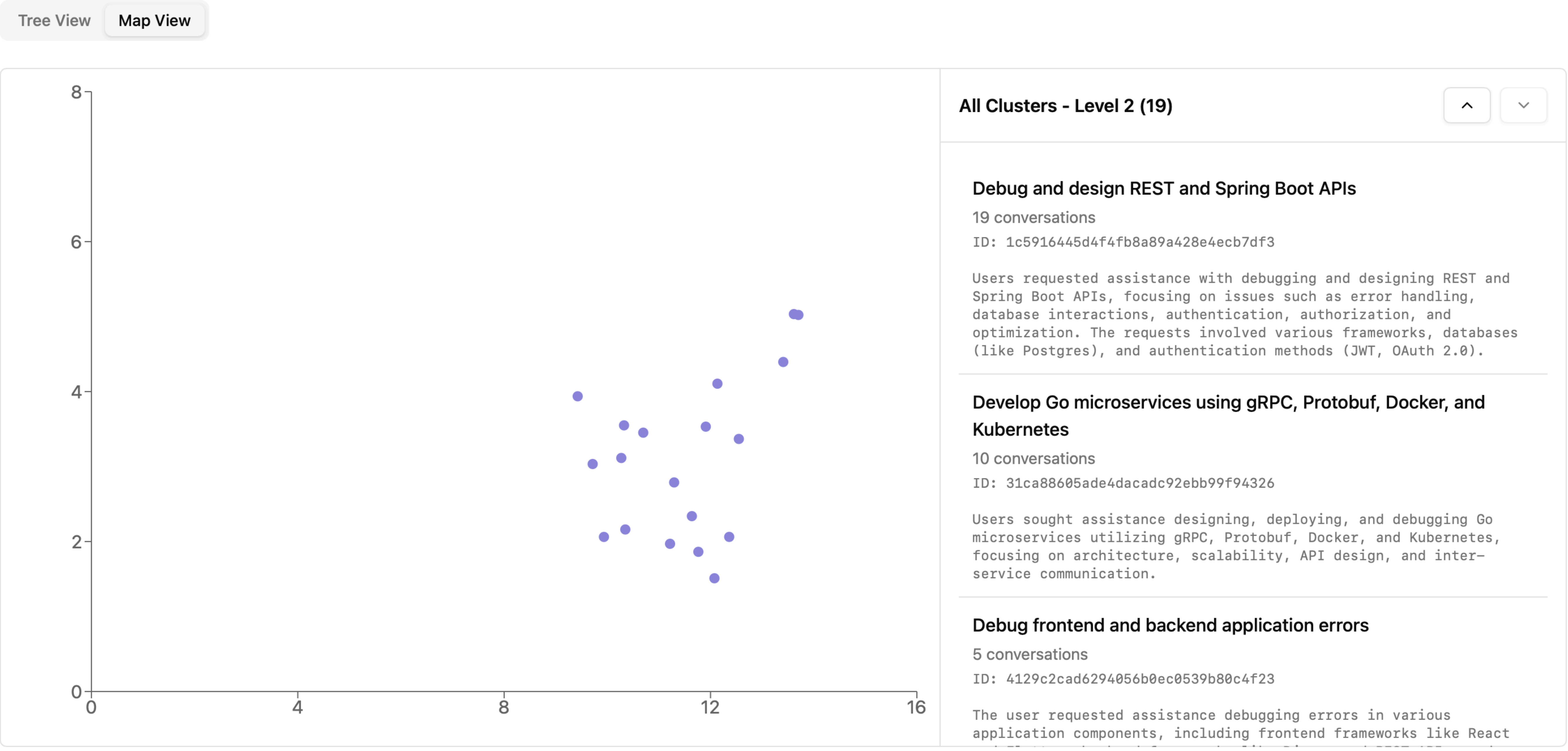
Synthetic Data

- We can generate multiple variations of the same query by varying tone, context etc
- This allows to validate base and simple cases
- Subsequently we blend in user data to generate queries that are on distribution
- With a large enough number of test cases, we can do sensitivity analysis and get a rough sense for model performance (https://ivanleo.com/blog/are-your-eval-improvements-just-pure-chance)

```
tests = [
    [
        "What is the average time to first response for issues in the azure
repository over the last 6 months? Has this metric improved or worsened?",
        [RunSQLReturnPandas],
    ],
    [
        "How many issues mentioned issues with Cohere in the 'vercel/next.js'
repository in the last 6 months?",
        [SearchIssues],
    ],
    [
        "What were some of the big features that were implemented in the last 4
months for the scipy repo that addressed some previously open issues?",
        [SearchSummaries],
    ],
 ]

for query, expected_result in tests:
    response = one_step_agent(query)
    for expected_call, agent_call in zip(expected_result, response):
        assert isinstance(agent_call, expected_call)
```

Query Analysis

- Eventually when we ship to production, we'll want to know how well we're performing for specific queries
- Topic modelling allows us to discover potential clusters/categories that are doing well or very bad



Debug and design REST and Spring Boot APIs

19 conversations

`ID: 1c5916445d4f4fb8a89a428e4ecb7df3`

Users requested assistance with debugging and designing REST and Spring Boot APIs, focusing on issues such as error handling, database interactions, authentication, authorization, and optimization. The requests involved various frameworks, databases (like Postgres), and authentication methods (JWT, OAuth 2.0).

Develop Go microservices using gRPC, Protobuf, Docker, and Kubernetes

10 conversations

`ID: 31ca88605ade4dacadc92ebb99f94326`

Users sought assistance designing, deploying, and debugging Go microservices utilizing gRPC, Protobuf, Docker, and Kubernetes, focusing on architecture, scalability, API design, and inter-service communication.

Debug frontend and backend application errors

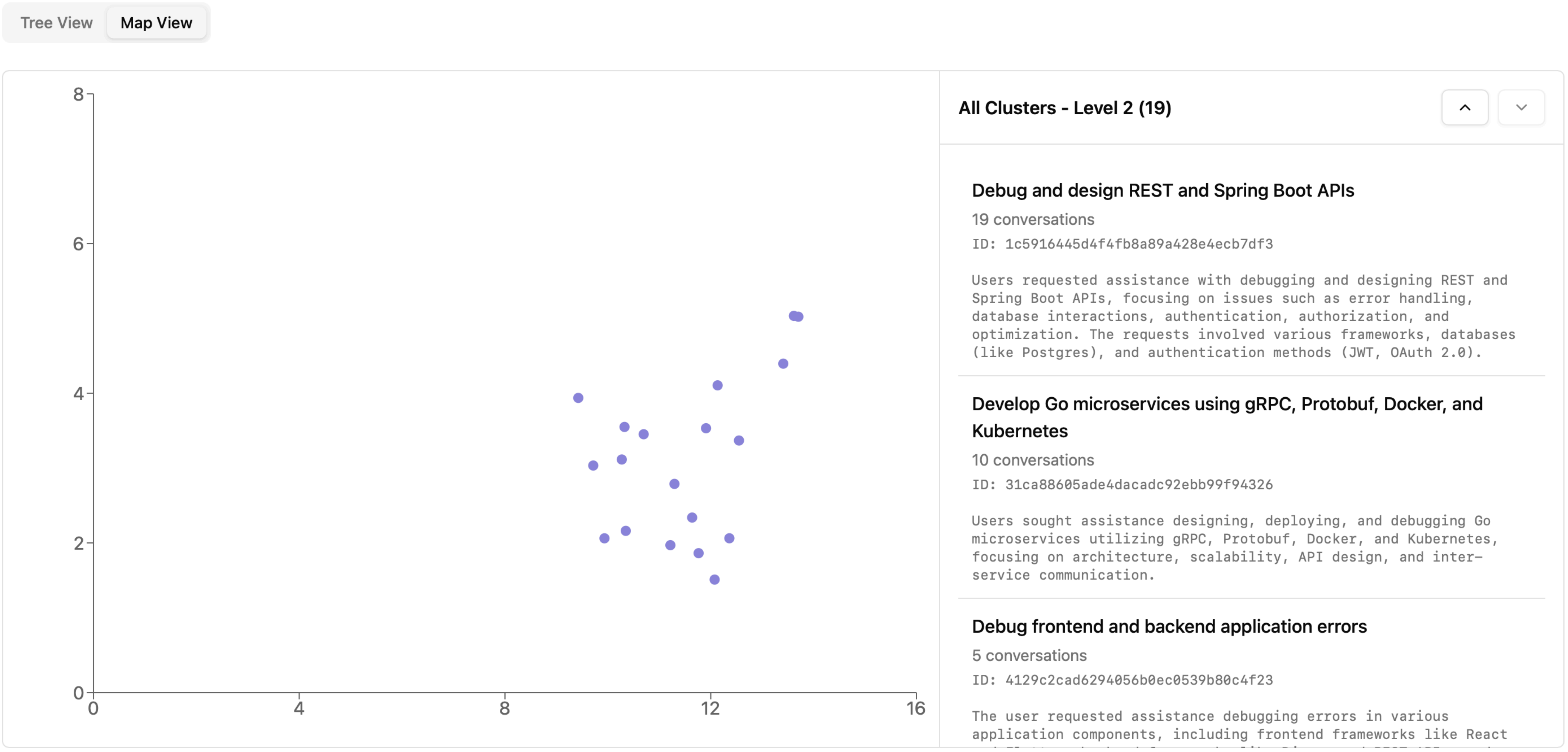
5 conversations

`ID: 4129c2cad6294056b0ec0539b80c4f23`

The user requested assistance debugging errors in various application components, including frontend frameworks like React

Query Analysis

- We want to then use these categories in production to segment populations of user queries to either highlight or prioritise
- Combine User Feedback with Volume etc - Segmentation is the name of the game



All Clusters - Level 2 (19)

Debug and design REST and Spring Boot APIs

19 conversations

`ID: 1c5916445d4f4fb8a89a428e4ecb7df3`

`Users requested assistance with debugging and designing REST and Spring Boot APIs, focusing on issues such as error handling, database interactions, authentication, authorization, and optimization. The requests involved various frameworks, databases (like Postgres), and authentication methods (JWT, OAuth 2.0).`

Develop Go microservices using gRPC, Protobuf, Docker, and Kubernetes

10 conversations

`ID: 31ca88605ade4dacadc92ebb99f94326`

`Users sought assistance designing, deploying, and debugging Go microservices utilizing gRPC, Protobuf, Docker, and Kubernetes, focusing on architecture, scalability, API design, and inter-service communication.`

Debug frontend and backend application errors

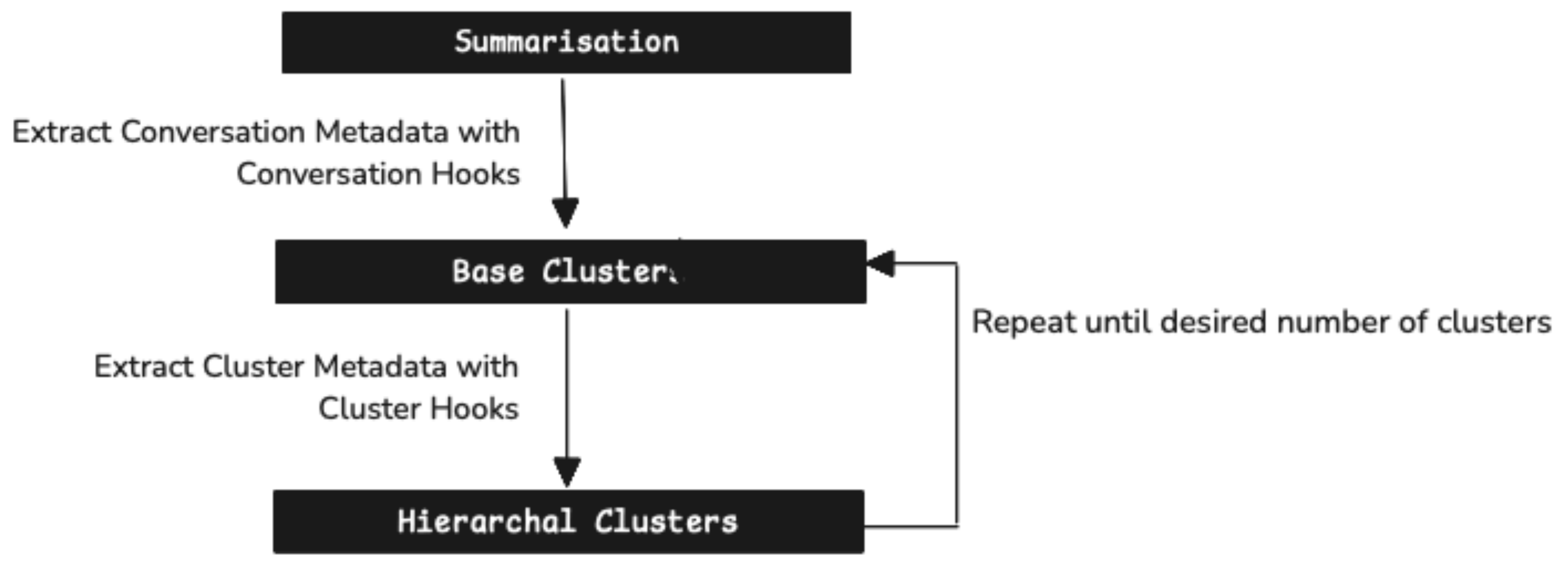
5 conversations

`ID: 4129c2cad6294056b0ec0539b80c4f23`

`The user requested assistance debugging errors in various application components, including frontend frameworks like React`

Kura

- Kura is a library that aims to help you generate explainable clusters
- We use a language model to extract general user requests from conversations
- We then generate clusters for these conversations and recursively combine them by embedding and summarising



Summarisation

Extract Conversation Metadata with Conversation Hooks

Base Clusters

Repeat until desired number of clusters

Extract Cluster Metadata with Cluster Hooks

Hierarchal Clusters

Kura

Validation using Synthetic Data

- We first start with high level categories that we'd like to work with

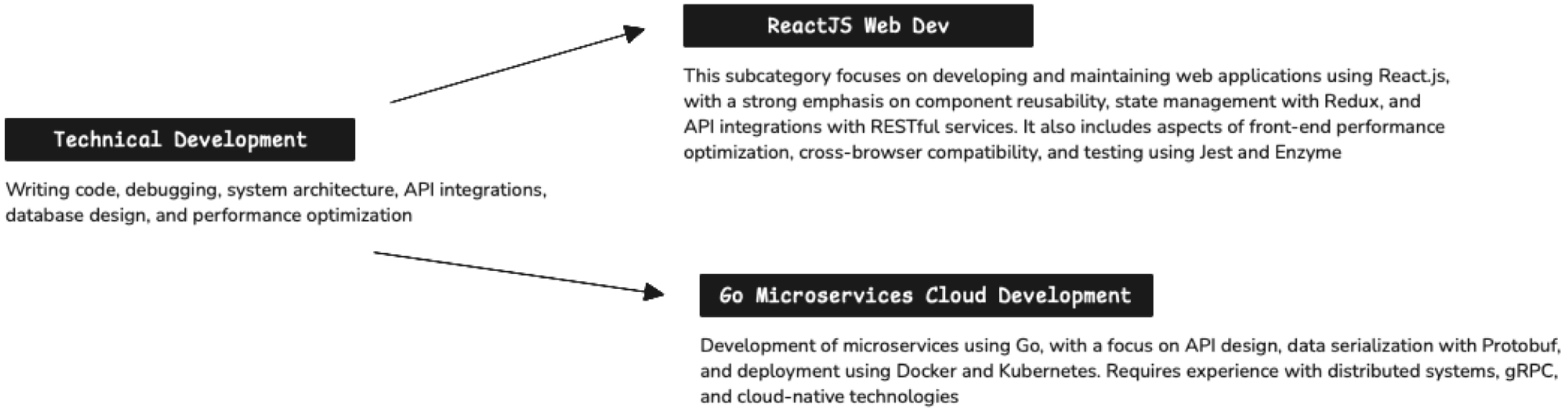
- Each has a category and a description

```
categories = [
    {
        "category": "Technical Development",
        "description": "Writing code, debugging, system architecture, API
integrations, database design, and performance optimization",
    },
    {
        "category": "Data Analysis & Visualization",
        "description": "Processing datasets, creating charts/graphs, statistical
analysis, and extracting insights from data",
    },
    {
        "category": "Content Creation",
        "description": "Writing documentation, blog posts, marketing copy, scripts,
stories, or other creative/professional writing",
    },
]
```

Kura

Validation using Synthetic Data

- Generate subcategories that are more specific and mention specific frameworks, details or use cases



Technical Development

Writing code, debugging, system architecture, API integrations, database design, and performance optimization

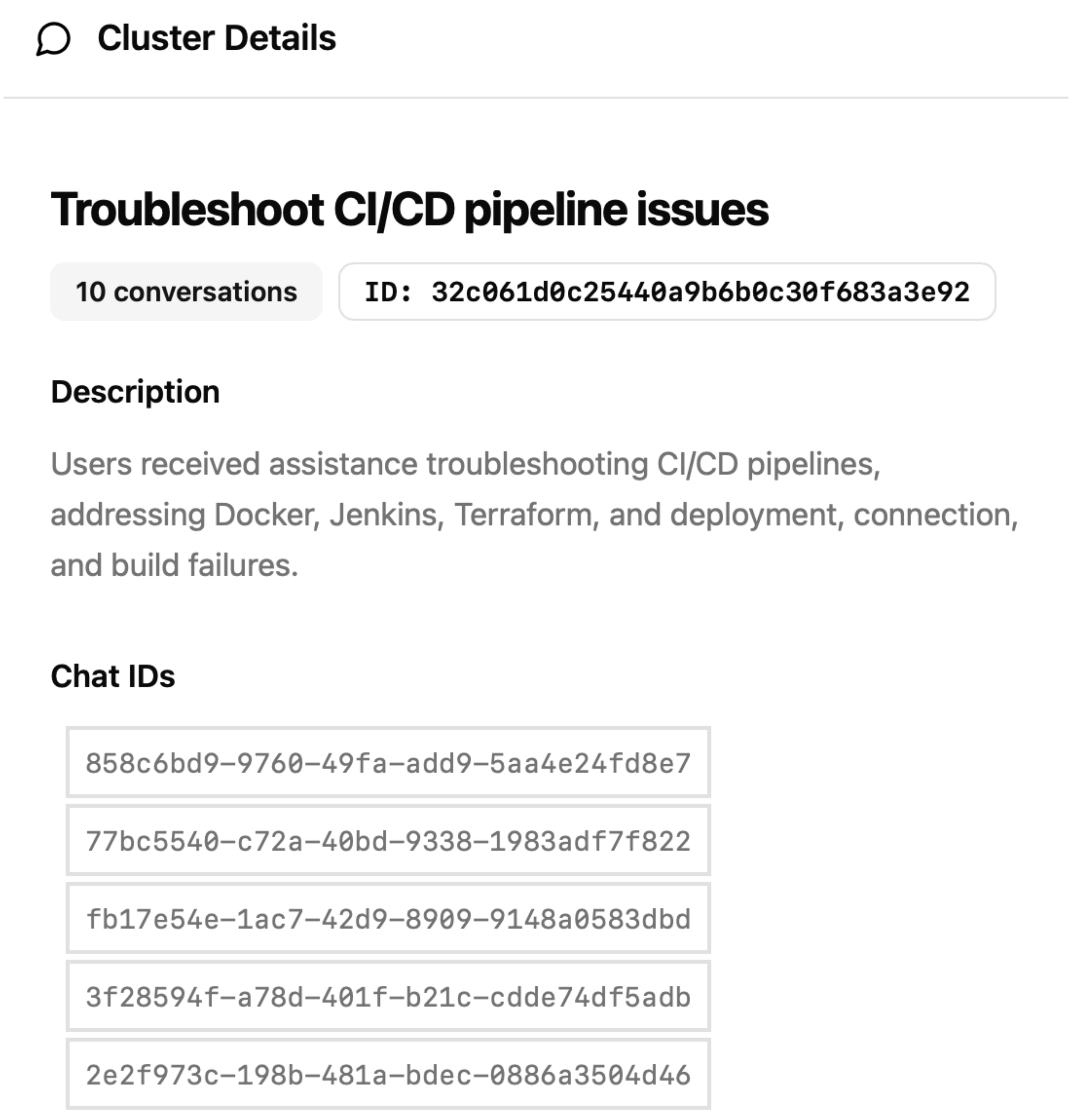
ReactJS Web Dev

This subcategory focuses on developing and maintaining web applications using React.js, with a strong emphasis on component reusability, state management with Redux, and API integrations with RESTful services. It also includes aspects of front-end performance optimization, cross-browser compatibility, and testing using Jest and Enzyme

Go Microservices Cloud Development

Development of microservices using Go, with a focus on API design, data serialization with Protobuf, and deployment using Docker and Kubernetes. Requires experience with distributed systems, gRPC, and cloud-native technologies

Kura



Troubleshoot CI/CD pipeline issues

10 conversations

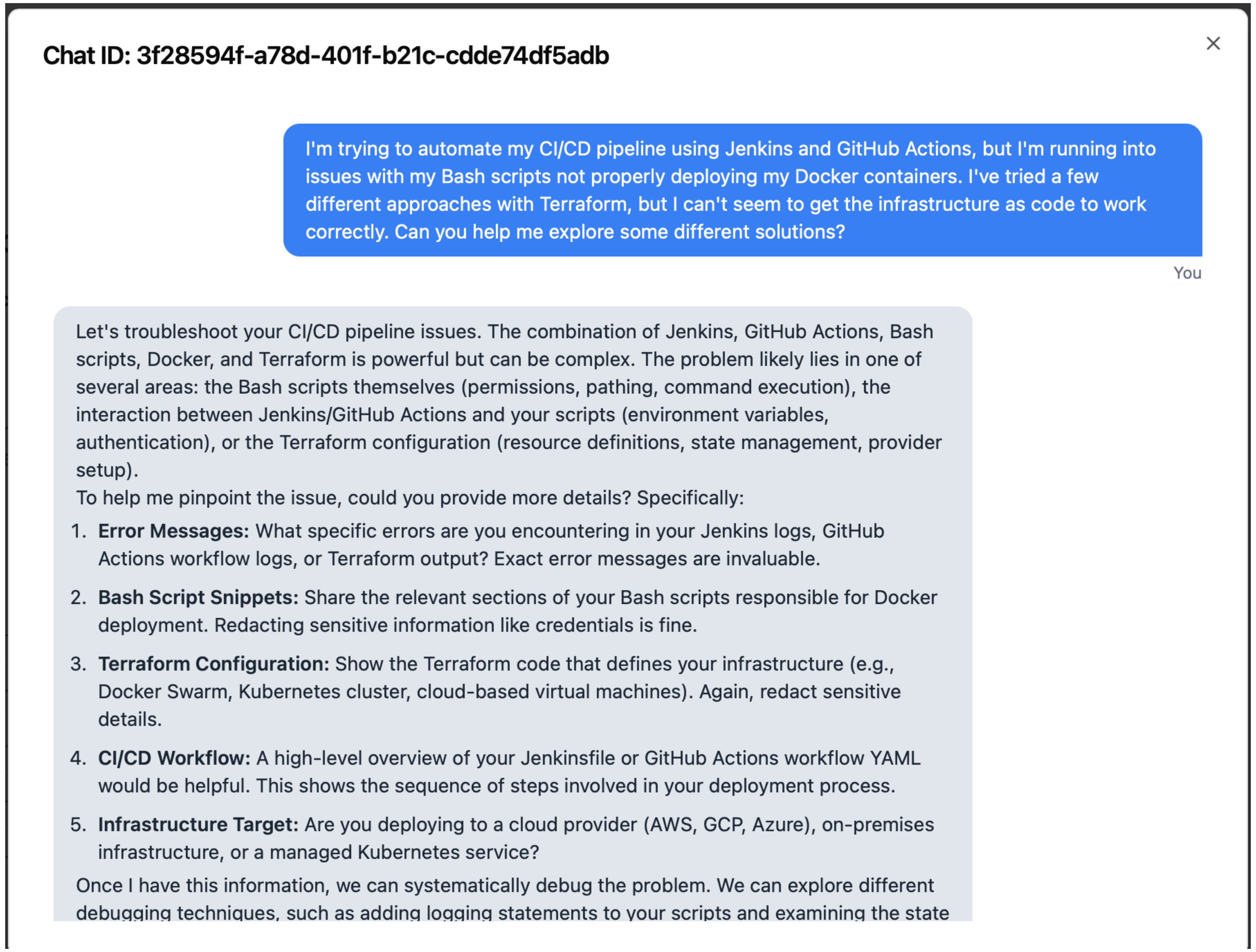
ID: 32c061d0c25440a9b6b0c30f683a3e92

Description

Users received assistance troubleshooting CI/CD pipelines, addressing Docker, Jenkins, Terraform, and deployment, connection, and build failures.

Chat IDs

- 858c6bd9-9760-49fa-add9-5aa4e24fd8e7
- 77bc5540-c72a-40bd-9338-1983adf7f822
- fb17e54e-1ac7-42d9-8909-9148a0583dbd
- 3f28594f-a78d-401f-b21c-cdde74df5adb
- 2e2f973c-198b-481a-bdec-0886a3504d46



Chat ID: 3f28594f-a78d-401f-b21c-cdde74df5adb

I'm trying to automate my CI/CD pipeline using Jenkins and GitHub Actions, but I'm running into issues with my Bash scripts not properly deploying my Docker containers. I've tried a few different approaches with Terraform, but I can't seem to get the infrastructure as code to work correctly. Can you help me explore some different solutions?

Let's troubleshoot your CI/CD pipeline issues. The combination of Jenkins, GitHub Actions, Bash scripts, Docker, and Terraform is powerful but can be complex. The problem likely lies in one of several areas: the Bash scripts themselves (permissions, pathing, command execution), the interaction between Jenkins/GitHub Actions and your scripts (environment variables, authentication), or the Terraform configuration (resource definitions, state management, provider setup).

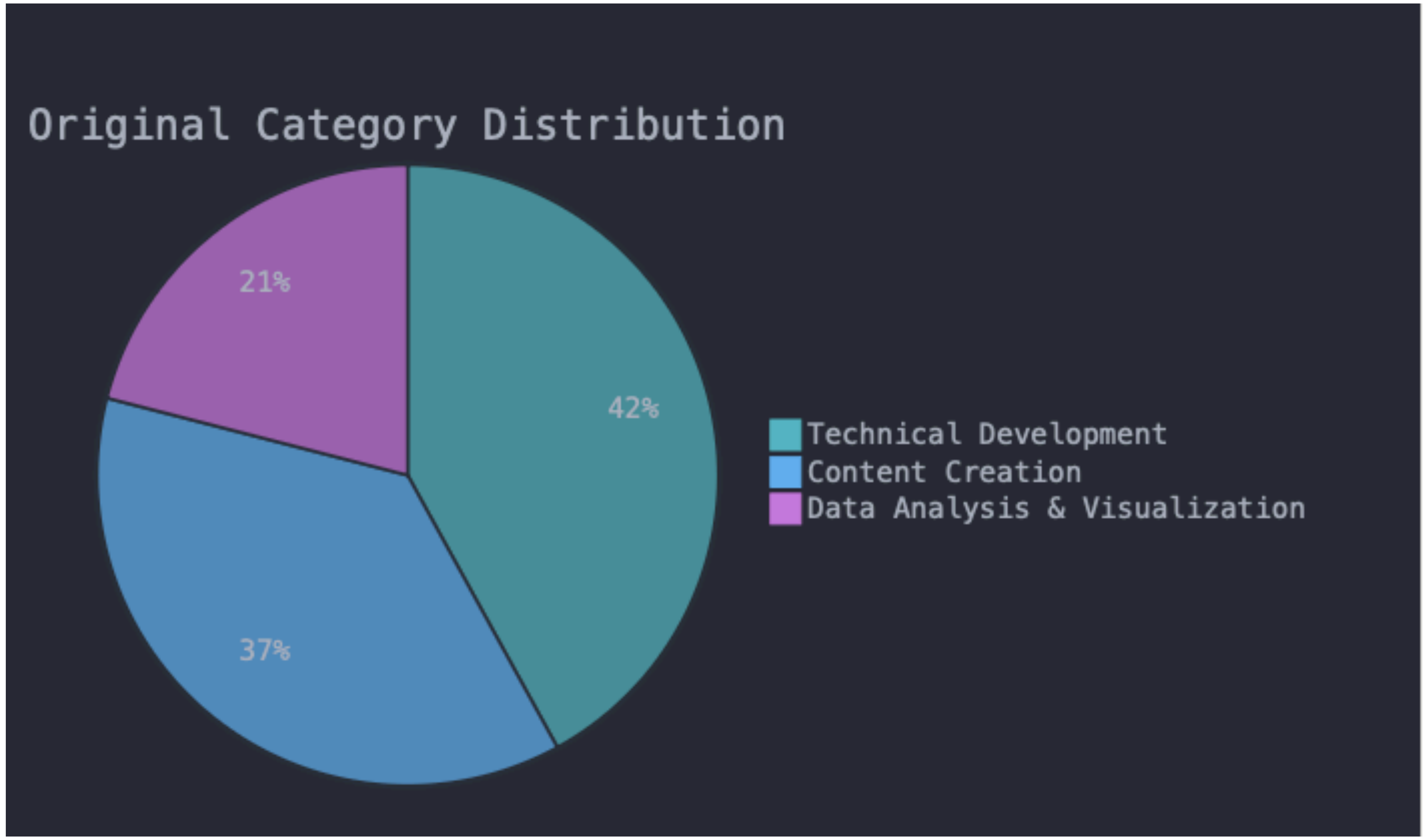
To help me pinpoint the issue, could you provide more details? Specifically:

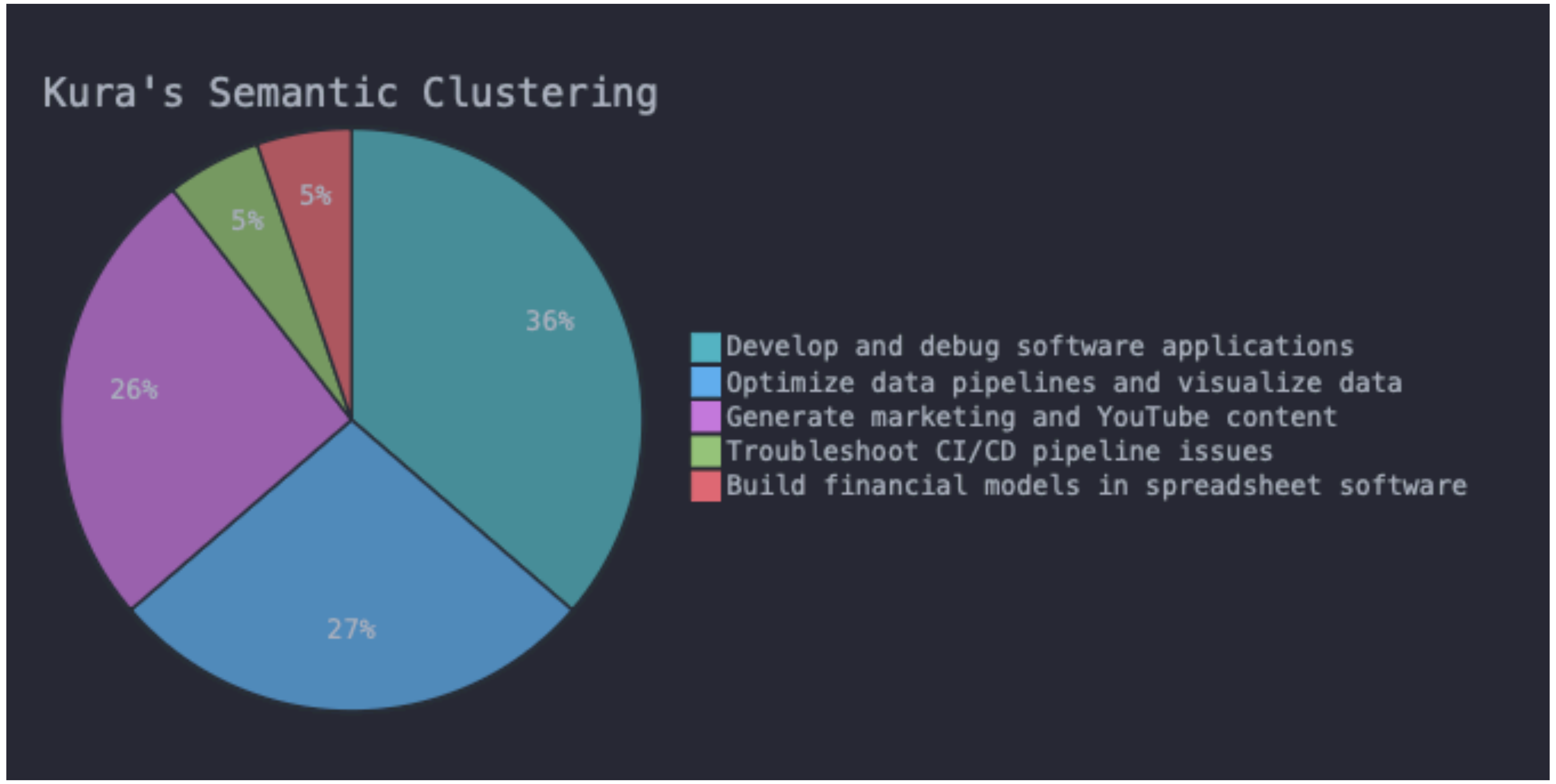
1. **Error Messages:** What specific errors are you encountering in your Jenkins logs, GitHub Actions workflow logs, or Terraform output? Exact error messages are invaluable.
2. **Bash Script Snippets:** Share the relevant sections of your Bash scripts responsible for Docker deployment. Redacting sensitive information like credentials is fine.
3. **Terraform Configuration:** Show the Terraform code that defines your infrastructure (e.g., Docker Swarm, Kubernetes cluster, cloud-based virtual machines). Again, redact sensitive details.
4. **CI/CD Workflow:** A high-level overview of your Jenkinsfile or GitHub Actions workflow YAML would be helpful. This shows the sequence of steps involved in your deployment process.
5. **Infrastructure Target:** Are you deploying to a cloud provider (AWS, GCP, Azure), on-premises infrastructure, or a managed Kubernetes service?

Once I have this information, we can systematically debug the problem. We can explore different debugging techniques, such as adding logging statements to your scripts and examining the state

Kura

- This was the original ground truth distribution that we started with
- Kura ended up categorising these into five distinct clusters that better reflect the technical nature of conversations





Kura

- You can give it a try today with docs at usekura.xyz
- I've written out two main ways
    - CLI with the command Kura that provides a simple react frontend to visualise clusters
    - Raw underlying Kura class to have more control

```
> pip install kura fastapi
> kura
INFO:     Started server process [41539]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press
CTRL+C to quit)
```

Query Understanding

Validation using Synthetic Data

- Fundamentally - these provide suggestions for how to think about and classify your conversations
- You can use other libraries like BERTopic that provide support for topic modelling across a larger time period for instance and experiment with different configurations
- The main goal is to basically learn how to prioritise and deprioritise features in response to user feedback or identify critical issues to resolve

Summary

What did we talk about

- Structured Outputs help eliminate a whole class of errors
- Binary metrics enable fast iteration
- Synthetic data helps find edge cases before production and even during deployment it helps us prevent regressions
- Query understanding and segmentation using libraries like Kura and Bertopic helps us to understand how users are using our application and the P(success | query type) for each individual type

Questions

Feel free to reach out!

- I’m perpetually online on twitter at ivanleomk so just message me
- I write longer form articles at ivanleo.com with a small newsletter that I’m experimenting with
- Do give instructor and kura a try! I’ll try my best to resolve issues and kura is very new so please bear with me as I work on it in between client work and other projects