MSIN0166 DATA ENGINEERING
INDIVIDUAL ASSIGNMENT

AI COMPETITOR INTELLIGENCE TOOL_

Candidate No: MXLJ4

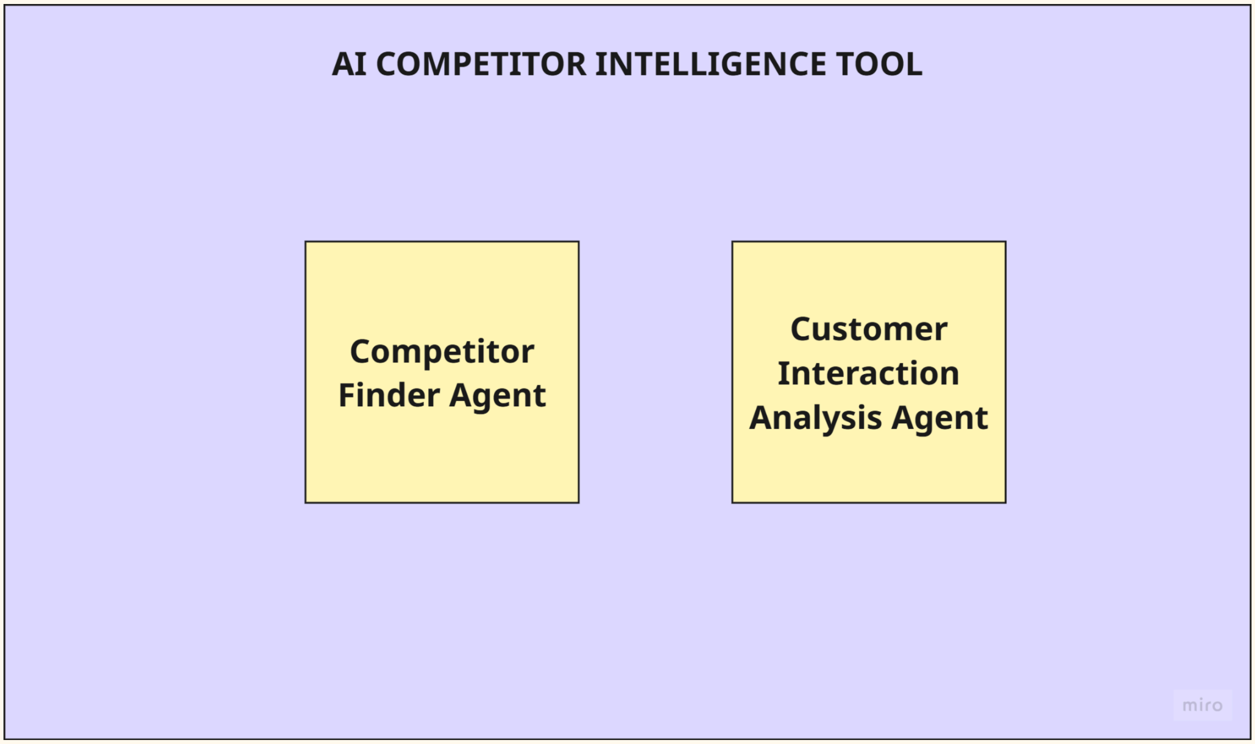
Word Count: 3955/4000

Introduction

This report details the design and implementation of an AI-Powered Competitor Intelligence Tool, which uses a dual-agent architecture to identify competitors and analyze customer interactions. It leverages LLMs and agentic systems to automate competitor intelligence gathering, addressing the need for efficient and insightful competitor analysis. The report also covers the system design, implementation using Streamlit and Agno framework, data preparation, adherence to software engineering best practices, and evaluation of the tool's performance and scalability.

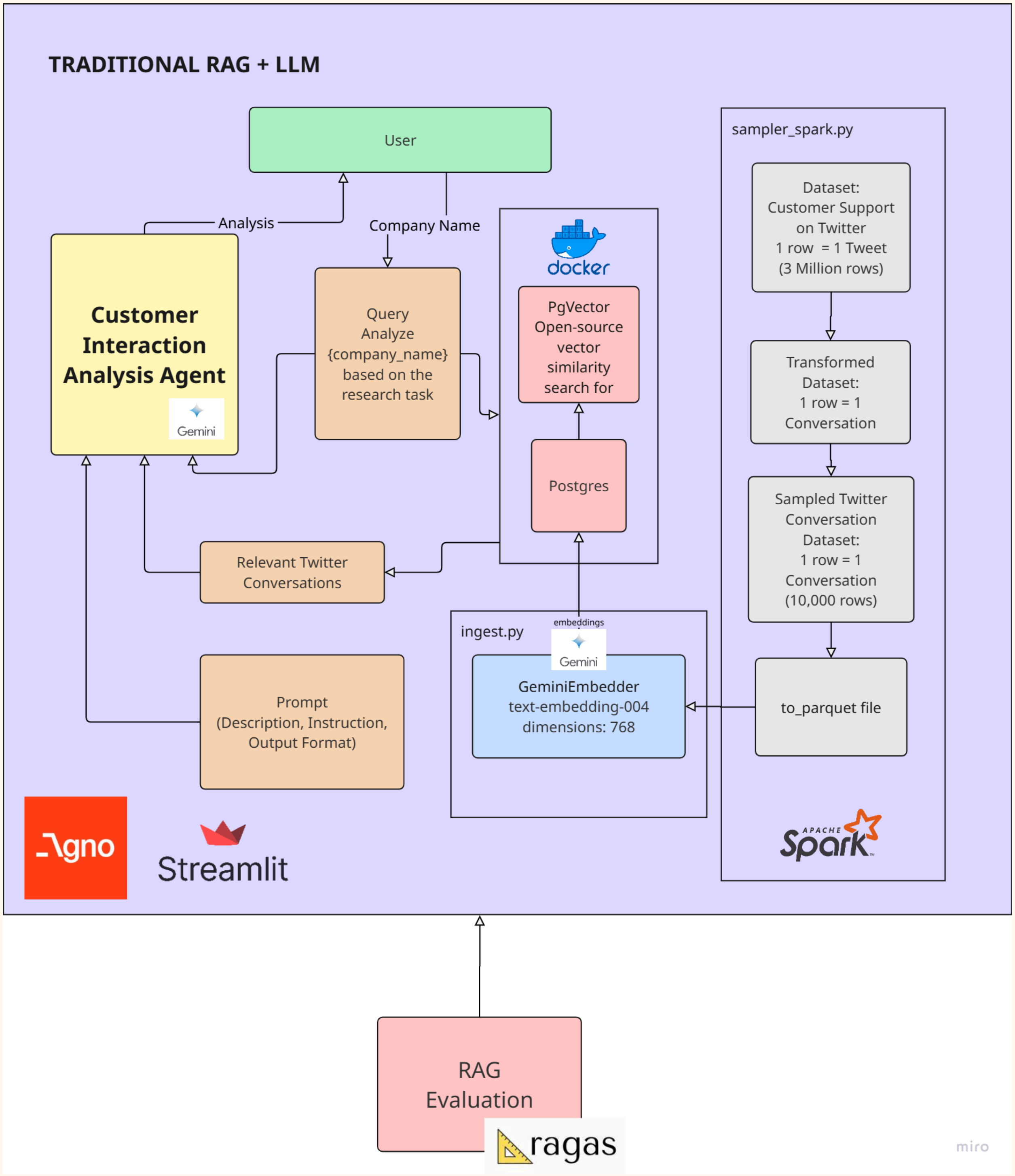
The full project code can be found at
https://github.com/prathamskk/ucl_rag_mcp_ai_competitor_intelligence_tool

System Design and Architecture



The AI Competitor Intelligence Tool is structured around a dual-agent architecture, comprising the Competitor Finder Agent and the Customer Interaction Analysis Agent. This design facilitates a modular approach, enabling focused functionality within each agent while allowing for seamless integration. The Competitor Finder Agent is responsible for identifying and profiling competitors, while the Customer Interaction Analysis Agent focuses on in-depth analysis of competitor behavior. This high-level architecture allows for scalability and adaptability, enabling the system to incorporate additional data sources and analytical capabilities in the future.

Customer Interaction Analysis Agent:



This component is designed to provide in-depth analysis of a specified company's customer interactions, leveraging publicly available Twitter data through a Retrieval-Augmented

Generation (RAG) pipeline. The design incorporates both an offline data preparation phase and an online, user-triggered analysis phase, as illustrated in the RAG + LLM diagram

Dataset Structure

The underlying dataset utilized for this preparation phase has the following characteristics:

Content: The source data is structured as a CSV file where each row represents an individual tweet. Conversations are implicitly defined by reply chains, and it's noted that meaningful conversations typically include at least one consumer request and one company response. The `inbound` field is key for identifying company user IDs and distinguishing customer messages from company replies.

Columns: The key columns include:

- `tweet_id`: A unique, anonymized identifier for the tweet, referenced by `response_tweet_id` and `in_response_to_tweet_id`.
- `author_id`: A unique, anonymized user identifier. Mentions (`@`) within tweet text are replaced with these anonymized IDs.
- `inbound`: A boolean flag indicating if the tweet is directed towards a company providing support, useful for organizing conversational data.
- `created_at`: Timestamp indicating when the tweet was posted.
- `text`: The actual content of the tweet. Sensitive information like phone numbers or email addresses has been masked (e.g., `__email__`).
- `response_tweet_id`: Comma-separated list of tweet IDs that are direct responses to the current tweet.
- `in_response_to_tweet_id`: The ID of the tweet to which the current tweet is a direct reply, if applicable.

Limitations:

- **Data Source**: Analysis is limited to Twitter data, which does not represent a competitor's entire online presence.
- **Brand Coverag**e: The analysis is limited to a select number of brands, based on the available Twitter dataset.
- **Data Age**: The system uses historical Twitter data (from 2017), which may not reflect current trends.

Future Expansion: The system architecture is designed with modularity in mind. This facilitates potential future enhancements, such as integrating live data streams from various sources, including other social media platforms or news feeds, to provide more real-time and comprehensive analysis.

Data Preparation

The data preparation phase begins with a substantial dataset comprising approximately 3 million tweets related to customer support interactions. A key challenge is transforming this raw tweet data into meaningful conversation threads suitable for analysis. This is handled by an Apache Spark job (`sampler_spark.py`). The logic identifies potential conversation starting points by selecting tweets that are marked as `inbound` (directed to a support account) and are not replies to other tweets (`in_response_to_tweet_id` is null). From this pool of initial tweets, a representative sample (e.g., 10,000) is randomly selected (using a fixed random state for reproducibility) to form the basis of the conversation dataset. For each selected starting tweet, the system reconstructs the full conversation thread by recursively following the reply chain using the `response_tweet_id` links, gathering all constituent tweets, and sorting them chronologically by `created_at`. This process transforms the data structure from individual tweets to complete conversation objects. These reconstructed conversations are then stored in the efficient Parquet format (`to_parquet file`). Following this, an ingestion script (`ingest.py`) processes these conversation objects. Crucially, each complete conversation is passed through an embedding model, specifically, Gemini's `text-embedding-004` (generating 768-dimension vectors), to capture its overall semantic meaning. These vector embeddings representing entire conversations are then stored and indexed in a specialized vector database – in this case, `PgVector`, an open-source extension for `Postgres`, which may be containerized using `Docker` for ease of deployment and management. This offline process ensures that the conversational data is readily available and optimized for fast retrieval during the analysis phase.

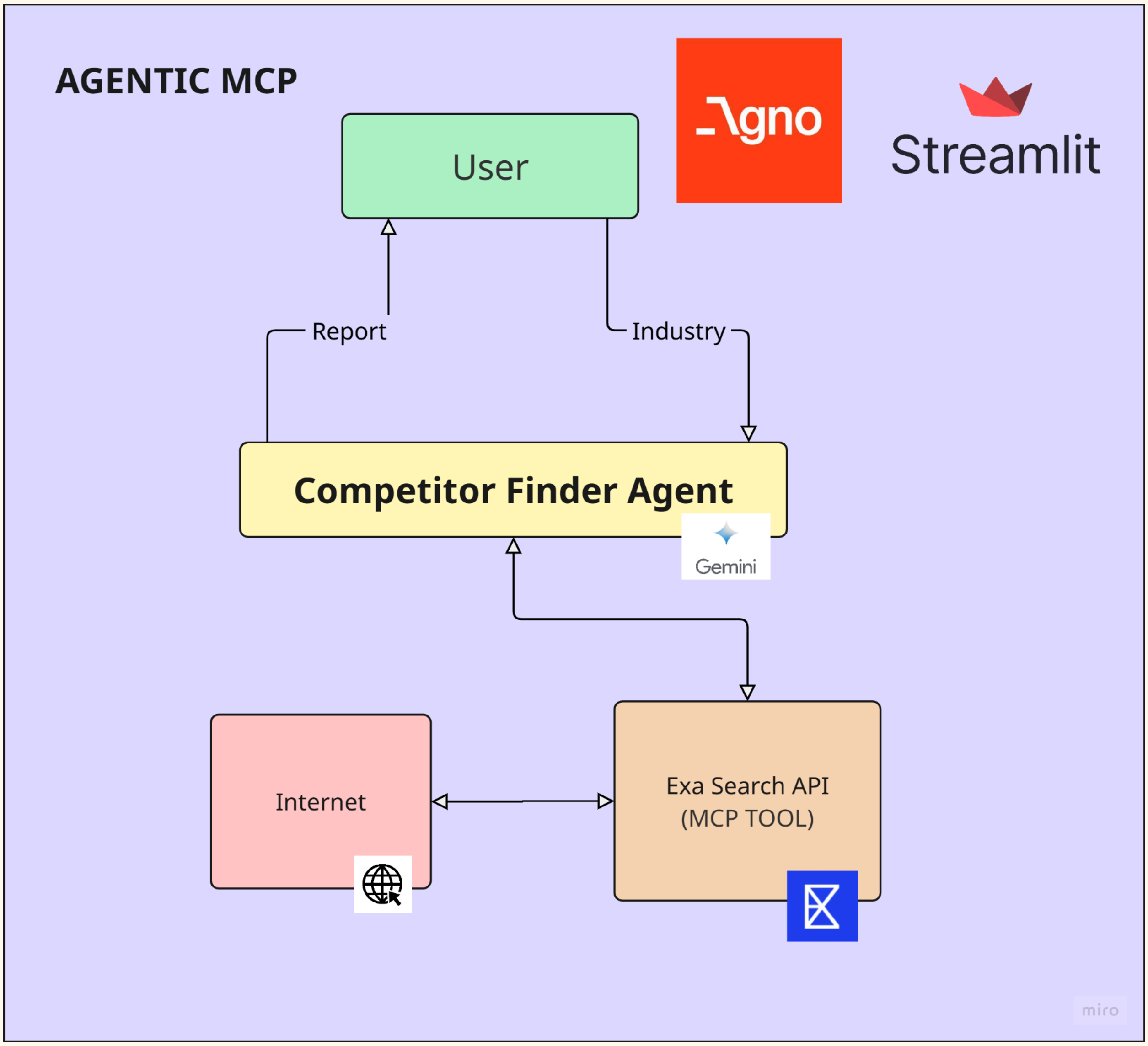
RAG Flow

The **RAG Flow** is initiated online when a user interacts with the Streamlit dashboard, providing a specific company name. The Customer Interaction Analysis Agent takes this input and formulates a query aimed at analyzing that company's interactions based on the prepared data. This query triggers a vector similarity search within the `PgVector` database to retrieve the most relevant Twitter conversations corresponding to the query's semantic content. These retrieved conversations serve as the context or "retrieved knowledge." This context is then dynamically combined with a carefully crafted prompt, which includes a description of the task, specific instructions for the analysis, and the desired output format. This augmented prompt, containing both the instructions and the relevant data, is sent to a Large Language Model (LLM), Gemini, for generation. The LLM synthesizes the information to produce a detailed analysis of the company's customer interactions based on the provided conversational evidence. Finally, this generated analysis is presented back to the user through the Streamlit interface.

RAG Evaluation

To ensure the quality and reliability of this RAG pipeline, the design includes provisions for **Evaluation** using frameworks like `ragas`. This allows for systematic assessment of the retrieval relevance and the quality of the generated analysis, facilitating iterative improvements to the prompt, retrieval strategy, or underlying models.

Competitor Finder Agent



Function and Input

The Competitor Finder Agent serves as a crucial component within the system. Its primary function is to identify and profile competitors operating within a specified industry. The agent receives the industry name as its input from the user.

Agentic Decision-Making

This agent operates with a degree of autonomy, making it an agentic entity. Based on its internal reasoning and the provided industry name, it will independently decide when and whether it needs to utilize external resources, such as the search API.

Leveraging Search API via MCP

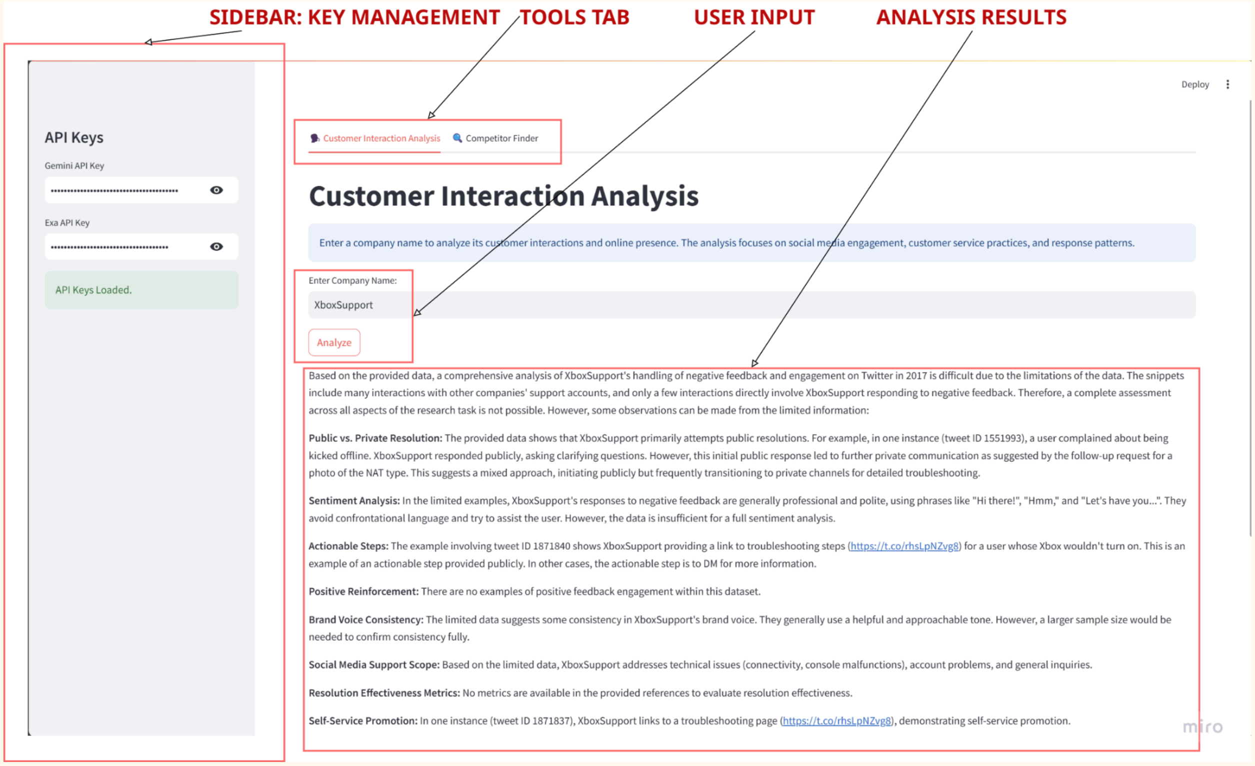
When the agent determines that gathering external information is necessary, it leverages the Exa Search API. This access is facilitated by the **Model Context Protocol (MCP)**. MCP functions as a standardized interface, enabling the agent, which is powered by Google's Gemini, to seamlessly connect to and utilize the Exa Search API as a pre-built integration.

Data Processing and Output

Once the agent has retrieved search results using the Exa Search API, Google's Gemini takes over to process the gathered information. It analyzes the data to identify the most relevant competitors within the specified industry. Finally, the agent generates a report as its output, which includes a list of identified competitors and potentially other pertinent information gathered during the process.

Implementation

Dashboard Overview



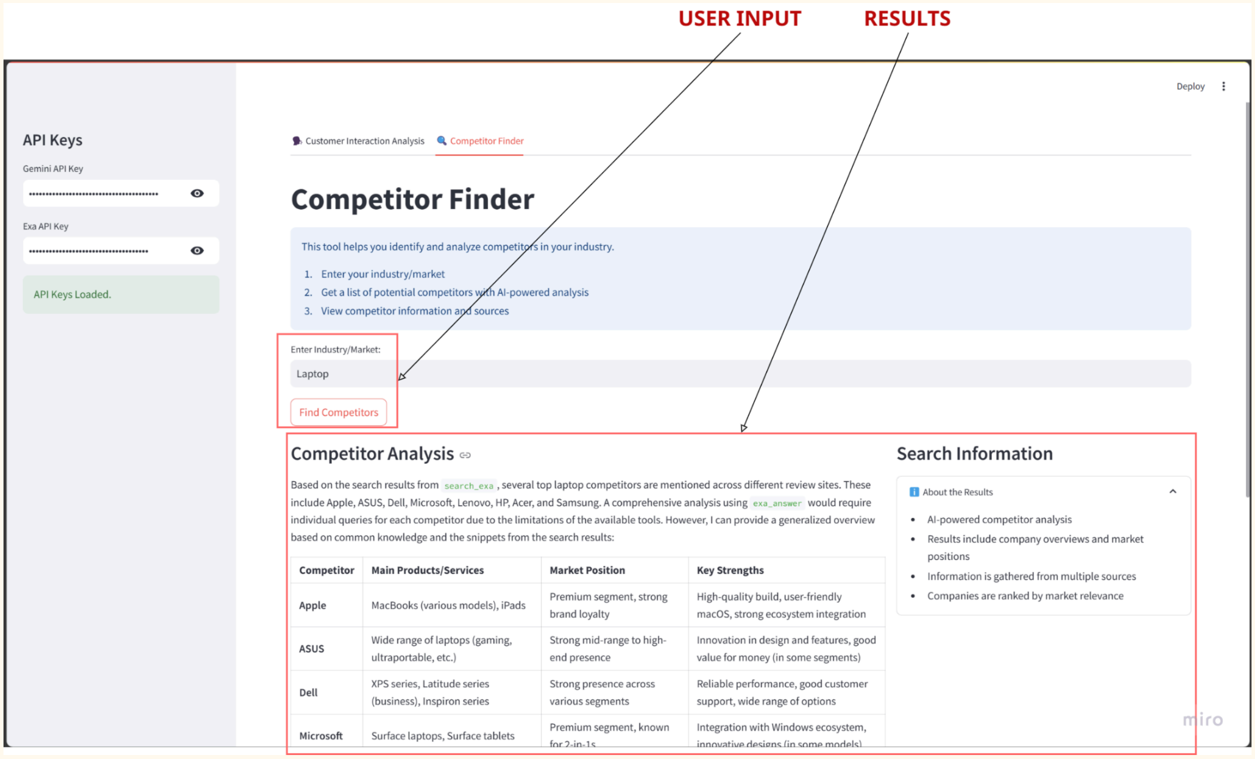
The project's dashboard was developed using Streamlit, an open-source Python framework designed for creating and sharing data-driven web applications with minimal effort. The dashboard's interface is organized into distinct sections:

Sidebar: Key Management:

The sidebar on the dashboard provides key management functionality, allowing users to input and manage their API keys. It includes input fields for both "Gemini API Key" and "Exa API Key", and masks the input for security purposes. Additionally, an "API Keys Loaded" indicator is displayed to confirm that the keys have been successfully entered.

Tools Tab:

The dashboard's main content area is organized into two separate tabs: "Customer Interaction Analysis" and "Competitor Finder". This tabbed layout allows users to easily switch between the two distinct analytical tools provided by the dashboard.



User Input:

Users have the ability to personalize their analysis by inputting specific parameters into the designated fields provided on the dashboard. Subsequently, they can initiate the analysis process by clicking the 'Analyze' button, which will then generate the corresponding results.

Analysis Results:

The results of the analysis are displayed in a text-based format on the dashboard. These text-based results encompass a range of insights and observations that are directly derived from the data.

Agents Implementation

The project utilizes Agno, an open-source platform for building, deploying, and monitoring AI agentic systems, to facilitate the analysis and response generation. Agno's model-agnostic design allows for the integration of various large language models, providing flexibility and scalability. The platform's capabilities, including memory management, external knowledge integration, and tool utilization, are leveraged to create high-performing agents.

Customer Interaction Analysis Agent

The following prompt was crafted to define the agent's role and core expertise, ensuring it operates within the intended analytical scope.

```
# Define agent description and instructions (keep existing ones)
agent_description = dedent("""\
    You are an expert analyst specializing in customer feedback and social media engagement.
    Your expertise includes:
    - Analyzing customer interactions on platforms like Twitter.
    - Identifying patterns in how companies respond to positive and negative feedback.
    - Assessing the effectiveness and professionalism of customer service practices on social media.
    - Extracting key insights and providing structured reports based on provided textual data.\
""")
```

This structured prompt was designed to guide the agent through a systematic process for analyzing customer interactions and extracting insights. Below are its operational phases.

```
agent_instructions = dedent(f"""\
    1. Information Extraction Phase 🔍
    - Carefully read the provided references (covering year 2017).
    - Identify and extract specific examples related to how {company_name} handles negative feedback and engages with the public on Twitter and dedicated customer service practices on social media.
    - Pay close attention to the specific aspects outlined in the research task (Public vs. Private Resolution, Sentiment Analysis, Actionable Steps, Positive Reinforcement, Brand Voice Consistency, Social Media Support Scope, Resolution Effectiveness Metrics, Self-Service Promotion, Personalized Communication).

    2. Analysis Phase 📊
    - For each of the aspects mentioned above, analyze the extracted information from the references.
    - Cross-reference information across different references if applicable.
    - Identify patterns, trends, and specific examples that illustrate {company_name}'s approach.
    - Note any inconsistencies or notable observations.

    3. Reporting Phase ✍️
    - Structure your analysis based on the points mentioned in the research task.
```

```
    - For each point, provide a concise summary of your findings,
supported by direct evidence or examples from the references.
    - Maintain an objective and analytical tone.

    4. Quality Control ✓
    - Ensure that all conclusions are directly supported by the provided
references.
    - Verify the accuracy of the extracted information.
    - Ensure clarity and coherence in your report.\
    """)
```

The following prompts were designed to guide the AI agent in generating structured and insightful analysis based on the retrieved data.

```
expected_output = dedent(f"""\
    Research Task: Based on the provided references (covering year 2017),
analyze how {company_name} handles negative feedback and engages with the
public on Twitter and dedicated customer service practices on social
media.

    Analyze their responses to negative comments, reviews, or complaints,
paying close attention to:

    * **Public vs. Private Resolution:** Determine the frequency with
which they publicly acknowledge issues versus offering private resolution,
according to the references.
    * **Sentiment Analysis (if your RAG can handle it):** Based on the
language used in their responses to negative feedback, assess the overall
professionalism and empathy.
    * **Actionable Steps:** Identify specific examples from the references
where {company_name} indicates steps being taken to address underlying
issues.
    * **Positive Reinforcement:** Analyze how they acknowledge and engage
with positive comments and praise, providing examples from the references.
    * **Brand Voice Consistency:** Describe the overall tone and voice
used in their public interactions, noting any inconsistencies observed in
the provided references.
    * **Social Media Support Scope:** Identify the range of customer
service inquiries addressed on social media, as evidenced in the
references.
    * **Resolution Effectiveness Metrics (if available in references):**
Based on the provided information, evaluate the clarity and effectiveness
of their resolution process, noting any metrics or indicators of success.
    * **Self-Service Promotion:** Identify instances where they direct
customers to FAQs or knowledge bases within the provided references.
    * **Personalized Communication:** Analyze the level of personalization
in their responses, providing examples from the references.
```

```
    **Instructions for RAG:** Your analysis must be solely based on the
information provided in the references. For each point, provide direct
evidence or examples from the text to support your conclusions.
""")
```

This section outlines the initialization and configuration of the AI agent, including embedding, vector database, and knowledge base integration. This implementation follows a traditional Retrieval-Augmented Generation (RAG) approach, where a query is used to retrieve relevant contextual data before being processed by an LLM for generation. The system integrates an offline data preparation phase and an online retrieval process to ensure the most relevant conversational data is available for analysis.

```
embedder = GeminiEmbedder(task_type="RETRIEVAL_QUERY",
api_key=_gemini_key)
vector_db = PgVector(table_name=_table_name, db_url=_db_url,
embedder=embedder, search_type=SearchType.hybrid)
knowledge_base = CSVKnowledgeBase(path=_csv_path, vector_db=vector_db ,
num_documents=10)

# Initialize Agent
agent = Agent(
    model=Gemini(id="gemini-1.5-flash", api_key=gemini_api_key_to_use),
    knowledge=knowledge_base,
    add_references=True,
    search_knowledge=True,
    show_tool_calls=True,
    description=agent_description,
    expected_output=expected_output,
    instructions=agent_instructions,
    markdown=True,
    debug_mode=True
)
```

Competitor Finder Agent

The **search prompt** guides the agent through a systematic process of identifying top competitors within a specified industry and analyzing their products, market position, and key strengths. This ensures that the retrieved data is relevant and presented in an organized manner.

```
search_prompt = f"""
1. Use search_exa to find top competitors in the {industry} industry
2. For each competitor found, use exa_answer to understand:
    - Their main products/services
    - Market position
    - Key strengths
3. Present the findings in a clear, structured format
"""
```

The **competitor agent** is built using the **Gemini model**, integrated with **ExaTools** for real-time search. It retrieves competitor insights through tool calls and structures the findings for clear interpretation.

```
competitor_agent = Agent(
model=Gemini(
    id="gemini-1.5-flash",
    api_key=st.session_state.gemini_api_key
),
tools=[ExaTools(
    api_key=st.session_state.exa_api_key,
    category="company",
    text_length_limit=1000,
)],
tool_call_limit=5,
show_tool_calls=True,
description="You are a competitive analysis expert. Your task is to find
and analyze relevant competitors.",
markdown=True,
debug_mode=True)
```

Data Preparation

This section outlines the steps taken to transform raw customer support interactions into structured conversational data optimized for retrieval and analysis.

Dataset Structure

The dataset consists of 2,811,774 rows and 7 columns, structured in CSV format. Each row represents an individual tweet, with attributes such as tweet IDs, author IDs, timestamps, and text.

Key data transformations include data types (e.g., user IDs) into numerical formats to optimize storage and processing efficiency.

	tweet_id	author_id	inbound	created_at	text	response_tweet_id	in_response_to_tweet_id
0	1	sprintcare	False	Tue Oct 31 22:10:47 +0000 2017	@115712 I understand. I would like to assist y...	2	3.0
1	2	115712	True	Tue Oct 31 22:11:45 +0000 2017	@sprintcare and how do you propose we do that	NaN	1.0
2	3	115712	True	Tue Oct 31 22:08:27 +0000 2017	@sprintcare I have sent several private messag...	1	4.0
3	4	sprintcare	False	Tue Oct 31 21:54:49 +0000 2017	@115712 Please send us a Private Message so th...	3	5.0
4	5	115712	True	Tue Oct 31 21:49:35 +0000 2017	@sprintcare I did.	4	6.0
5	6	sprintcare	False	Tue Oct 31 21:46:24 +0000 2017	@115712 Can you please send us a private messa...	5,7	8.0
6	8	115712	True	Tue Oct 31 21:45:10 +0000 2017	@sprintcare is the worst customer service	9,6,10	NaN
7	11	sprintcare	False	Tue Oct 31 22:10:35 +0000 2017	@115713 This is saddening to hear. Please shoo...	NaN	12.0
8	12	115713	True	Tue Oct 31 22:04:47 +0000 2017	@sprintcare You gonna magically change your co...	11,13,14	15.0
9	15	sprintcare	False	Tue Oct 31 20:03:31 +0000 2017	@115713 We understand your concerns and we'd l...	12	16.0

Data Processing

To extract meaningful conversations, an **Apache Spark** job filters inbound tweets, reconstructs reply chains, and organizes complete conversations. These are stored in **Parquet** format for efficient retrieval.

Starting Tweet Selection Logic

The system identifies potential conversation starting points using the following criteria:

- **Inbound Tweets Only:** Tweets must be marked as "inbound" (i.e., directed to a company support account).
- **Not a Reply:** The tweet must not be a response to another tweet (`in_response_to_tweet_id` is `NULL`).
- **Random Sampling for Efficiency:** A subset of tweets (e.g., 10,000) is randomly selected to form the conversation dataset, ensuring reproducibility by using a fixed random seed.

Reply Chain Construction

Once starting tweets are selected, the full conversation thread is reconstructed using these steps:

1. **Follow Response Links:** The system recursively follows `response_tweet_id` values to gather replies.

2. **Sort by Timestamp:** The collected tweets are sorted chronologically using the `created_at` field to maintain conversation flow.
3. **Store as Conversation Objects:** The reconstructed threads are stored as structured conversation objects for downstream processing

Example of a Retrieved Reply Chain

The screenshot below showcases an example of a reconstructed reply chain. Each conversation begins with an inbound customer query, followed by responses from the company or other users.

	tweet_id	author_id	inbound	created_at	text	response_tweet_id	in_response_to_tweet_id
2640273	2812607	784256	True	2017-11-27 01:12:40+00:00	@AppleSupport I am TRYING to download apps wit...	[2812606]	NaN
2640272	2812606	AppleSupport	False	2017-11-27 01:20:00+00:00	@784256 We want to help! What device are you u...	[2812604]	2812607.0
2640271	2812604	784256	True	2017-11-27 01:23:34+00:00	@AppleSupport iPhone 8+. It isn't saying an er...	[2812602]	2812606.0
2640268	2812602	AppleSupport	False	2017-11-27 01:32:50+00:00	@784256 How many bars of signal strength do yo...	[2812603]	2812604.0
2640269	2812603	784256	True	2017-11-27 01:33:21+00:00	@AppleSupport Right now I have 3/4 on LTE	[2812605]	2812602.0
2640270	2812605	AppleSupport	False	2017-11-27 01:47:00+00:00	@784256 Perfect. We'd like to investigate furt...	NaN	2812603.0

The table below represents the final structured format of the customer support conversations before embedding generation. Each row corresponds to a reconstructed conversation, uniquely identified by a **conversation_id**. The table includes key metadata such as **start_time** and **end_time** to indicate the duration of the conversation, along with **number_of_turns**, which quantifies the number of exchanges. Additionally, the **tweets** and **tweet_ids** columns aggregate all messages within the conversation thread, preserving the full interaction history.

	conversation_id	start_time	end_time	number_of_turns	tweets	tweet_ids
0	723457c6-caab-4156-983d-76e9a62faf20	2017-11-27 01:14:51+00:00	2017-11-27 01:46:29+00:00	4	[{'tweet_id': 2812638, 'author_id': '142760', ...	[2812638, 2812636, 2812637, 2812639]
1	138a4725-55c0-4636-b68d-76495ce0a85d	2017-10-06 20:13:45+00:00	2017-10-13 16:51:33+00:00	5	[{'tweet_id': 1084025, 'author_id': '118189', ...	[1084025, 1084022, 1084024, 1084023, 1084021]
2	9877f91a-b1b7-43a7-bbf5-bb6320bfe4ca	2017-11-22 22:37:00+00:00	2017-11-22 22:41:20+00:00	2	[{'tweet_id': 656755, 'author_id': '276284', '...	[656755, 656754]
3	3f04444d-7f49-4278-ad67-06e337585195	2017-10-11 05:26:07+00:00	2017-10-11 08:28:47+00:00	2	[{'tweet_id': 728827, 'author_id': '294581', '...	[728827, 728826]
4	eb519b2d-b73f-47ad-a434-431abf2f4e3b	2017-11-19 19:55:53+00:00	2017-11-19 20:02:30+00:00	2	[{'tweet_id': 2698968, 'author_id': '758288', ...	[2698968, 2698967]

This structured format enables efficient retrieval and analysis of customer interactions. The dataset was then exported to **Parquet** format for optimized storage and retrieval.

Vector Embedding and Storage

To facilitate efficient retrieval and clustering, each reconstructed conversation is transformed into a high-dimensional vector representation using Gemini's **text-embedding-004** model. The embedding model generates **768**-dimensional vectors. These vector representations are then stored in **PgVector**. The task type was set to "**CLUSTERING**", enabling the system to group similar conversations based on their contextual meaning.

Software Engineering Best Practices

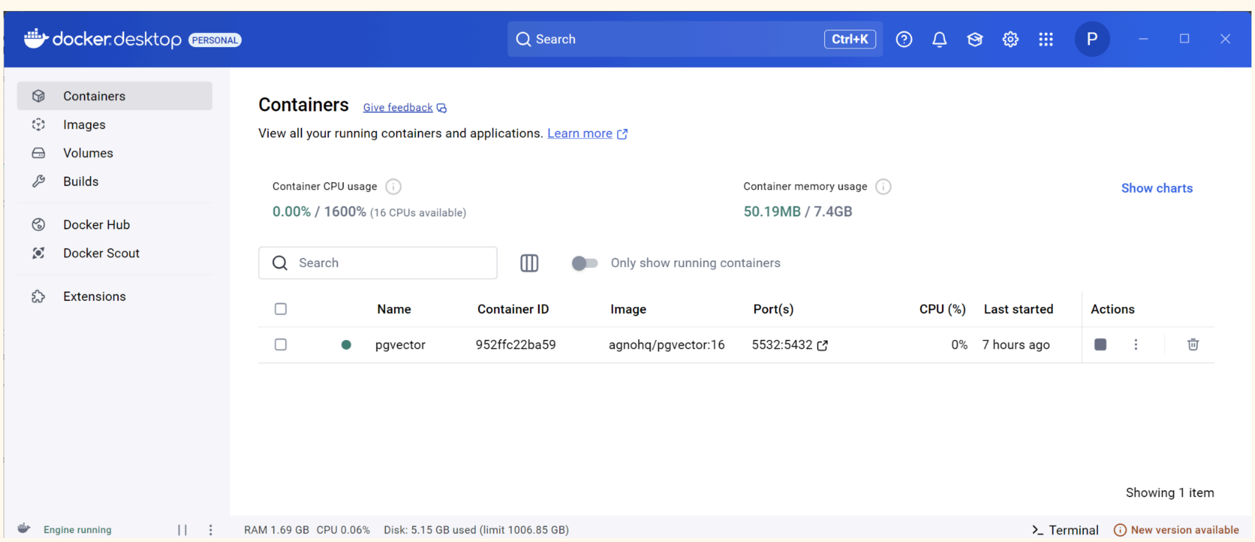
Docker Containers – Environment Consistency and Scalability

Docker was used to containerize the application, ensuring a consistent runtime environment across different systems. It also hosts PostgreSQL with the **pgvector** extension, enabling scalable and efficient vector search capabilities.

run_pgvector.sh

```
#!/bin/bash

docker run -d -e POSTGRES_DB=ai -e POSTGRES_USER=ai -e
POSTGRES_PASSWORD=ai -e PGDATA=/var/lib/postgresql/data/pgdata -v
pgvolume:/var/lib/postgresql/data -p 5532:5432 --name pgvector
agnohq/pgvector:16
```



Dockerfile

```
FROM python:3.11-slim
```

```
WORKDIR /app
COPY . .
RUN pip3 install -r requirements.txt
EXPOSE 8501
HEALTHCHECK CMD curl --fail http://localhost:8501/_stcore/health
ENTRYPOINT ["streamlit", "run", "main.py", "--server.port=8501",
"--server.address=0.0.0.0"]
```

UV - Project Management

UV was used for **package and project management**, handling dependencies efficiently with a universal lockfile. It provides a fast and streamlined alternative to traditional Python package managers, ensuring consistency and performance in project environments.

pyproject.toml

```
[project]
name = "ucl-rag-mcp-ai-competitor-intelligence-tool"
version = "0.1.0"
description = "Add your description here"
readme = "README.md"
requires-python = ">=3.11"
dependencies = [
    "agno>=1.2.5",
    "exa-py==1.7.1",
    "google-genai==1.8.0",
    "pgvector>=0.4.0",
    "psycopg2>=2.9.10",
    "pyarrow>=19.0.1",
    "python-dotenv>=1.1.0",
    "ragas>=0.2.14",
    "sqlalchemy>=2.0.40",
    "streamlit==1.41.1",
]
```

Environment Files - Secure credentials management

Environment files (**.env**) were used for securely managing sensitive credentials and configuration variables. This ensures separation of secrets from code, enhancing security and simplifying deployment across different environments.

example .env file (sample)

```
GEMINI_API_KEY="xxxxxxx"
DB_URL="postgresql+psycopg2://xxxx:xxxxx@localhost:5532/xxxxx"
TABLE_NAME="name_of_your_table"
CSV_FILE_PATH="xxxxx"
```

```
EXA_API_KEY="xxxxx"
OPENAI_API_KEY="xxxx"
```

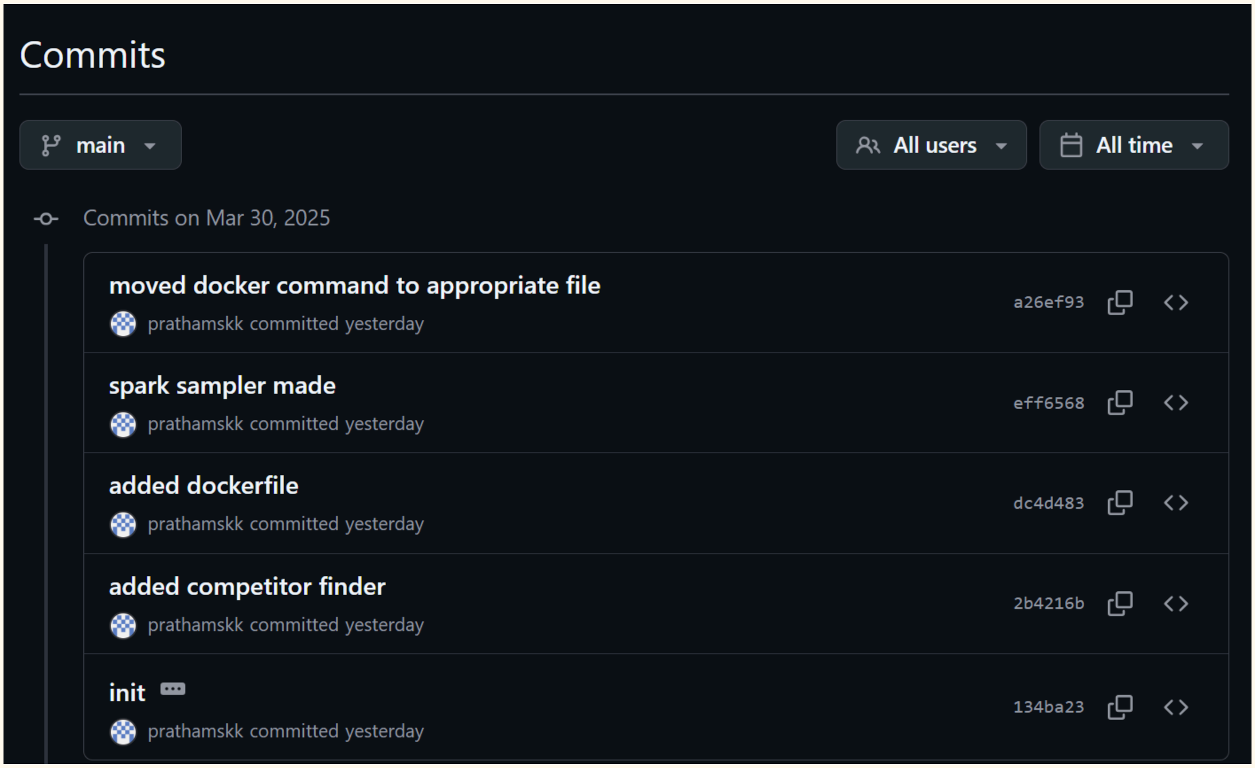
Logging - Reliability

Detailed logs were maintained to track every detail, revisit past system activities, and diagnose why certain components may not be functioning properly.

```
DEBUG ***********************************************************  METRICS  ***********************************************************
DEBUG * Tokens:                      input=2961, output=139, total=3100
DEBUG * Time:                        1.3519s
DEBUG * Tokens per second:           102.8166 tokens/s
DEBUG * Time to first token:         0.2398s
DEBUG ***********************************************************  METRICS  ***********************************************************
DEBUG --------------------------------------------------- Google Response Stream End --------------------------------------------------
DEBUG Added 4 Messages to AgentMemory
DEBUG Added AgentRun to AgentMemory
DEBUG Logging Agent Run
DEBUG ************************************** Agent Run End: cd736818-df81-448e-a3f5-99f4ca65f6c2 **************************************
DEBUG Function: search_exa registered with exa
DEBUG Function: get_contents registered with exa
DEBUG Function: find_similar registered with exa
DEBUG Function: exa_answer registered with exa
DEBUG ***************************************** Agent ID: d35dbafa-9ac2-4b0e-ab91-bc20689667d6 ****************************************
DEBUG **************************************** Session ID: 3a4c919f-1502-4f9b-aba6-67b24c157342 ***************************************
DEBUG ************************************* Agent Run Start: 7ad0b2a5-ebfb-4a61-b8b3-25bff3302deb *************************************
DEBUG Processing tools for model
DEBUG Included function search_exa from exa
DEBUG Included function get_contents from exa
DEBUG Included function find_similar from exa
DEBUG Included function exa_answer from exa
DEBUG -------------------------------------------------- Google Response Stream Start -------------------------------------------------
DEBUG ---------------------------------------------------- Model: gemini-1.5-flash ----------------------------------------------------
DEBUG ============================================================= system ============================================================
```

GIT - version control system

Git was used to track code changes and maintain version history.



Evaluation

This section evaluates the performance, effectiveness, scalability, maintainability, and limitations of the developed AI-Powered Competitor Intelligence Tool, reflecting on the challenges encountered and lessons learned during the project.

Performance and Effectiveness

The system's processing speed was satisfactory, allowing for efficient operation and analysis. The Customer Interaction Analysis Agent consistently provided accurate and valuable insights by effectively retrieving and processing pertinent information. This proved to be a valuable asset in understanding customer interactions and improving overall customer experience.

However, the Competitor Finder Agent's performance was less consistent. While it did provide some useful competitor intelligence, its unreliability and tendency to produce inconsistent results detracted from its overall effectiveness and limited the tool's overall value to the business. This inconsistency made it challenging to rely solely on the Competitor Finder Agent for accurate and comprehensive competitor analysis, potentially hindering strategic decision-making and competitive positioning.

RAG Pipeline Evaluation (RAGAS)

The RAG pipeline of the Customer Interaction Analysis agent was evaluated using the ragas library. Key retrieval metrics, context_precision and context_recall, were calculated using evaluation questions based on the Twitter conversation data.

Context_precision measures if retrieved information was relevant to the query. Context_recall measures if all relevant information was retrieved.

```python
# Example evaluation data using real conversation examples
evaluation_data = {
    "question": [
        "How does AppleSupport handle negative customer feedback on Twitter?",
        "What are the key aspects of AppleSupport's social media customer service?",
        "How does AppleSupport promote self-service options to customers?",
    ],
    "answer": [
        "AppleSupport typically responds to negative feedback professionally and promptly on Twitter, often directing users to private channels for detailed support while acknowledging issues publicly.",
        "AppleSupport's social media customer service focuses on prompt responses, professional tone, and directing users to appropriate support channels while maintaining brand voice consistency.",
        "AppleSupport promotes self-service through their support website, directing users to FAQs, knowledge base articles, and automated support options when appropriate.",
    ],
    # Using real reference texts from the conversation data
    "reference": [
        """@731750 When I long press on my app I don't get the selection to delete them, the apps "shake" but no red lines pop up to delete any of the apps
        @AppleSupport We'd be happy to see what's causing you to be unable to delete those apps. Let's start by restarting the iPhone and see if that helps at all.
        @731750 I've restarted and made my phone is updated to the latest version, still nothing.
        @AppleSupport Do you have Restricitions turned on in Settiings > General?
        @731750 yup! user error 😔 dont even know how that got turn on. Thanks for the help i really appreciate your responses.""",

        """@369384 is it 2 much to ask for my devices to work properly? why am I buying products that are 1k, outdated 1 year & unusable in 5?
        @AppleSupport If you're having issues with your Apple products, we're here for you. Send us a DM with the details; we'll go from there.
        @369384 #iphone #MacBookPro #ipad-air useless 🍎apps & each update insures my dependable 3rd party apps don't work! 1/3 of memory is used by 🍎 trash
        @AppleSupport We want to take a closer look at this with you. Reach out via Direct Message using the following link: https://t.co/GDrqU22YpT""",

        """@405403 is there away to cancel preorder on iTunes from iPad or iPhone
        @AppleSupport Great, question! You can only manage your pre-orders in iTunes on a Mac or PC: https://t.co/df4pj0iEqO
        @405403 and u can't call in and cancel it I tried
        @405403 u guys need to make it where we can its BS
        @AppleSupport You can definitely reach out to our iTunes Store support team here if you'd like: https://t.co/SDIe7UiyJN"""
    ]
}
```

```
Evaluating: 100%|██████████████████████████████| 6/6 [02:41<00:00, 26.96s/it]

Evaluation Results:
{'context_precision': 0.8, 'context_recall': 0.94}
```

The evaluation yielded a context_precision score of 0.80 and a context_recall score of 0.94. The former indicates that most retrieved information was relevant, and the latter shows the system effectively found almost all necessary context.

These scores demonstrate a strong retrieval component, supporting the agent's effectiveness in generating consistent and relevant analyses. While these metrics provide valuable insights into retrieval quality, a comprehensive RAG evaluation could include metrics assessing generation quality, such as faithfulness and answer_relevancy.

Scalability and Maintainability

Scalability: The system's containerized architecture and PgVector enable good scalability. However, significant increases in data volume, user load, or companies analyzed would likely encounter API, computational, and database bottlenecks. Scaling to diverse, real-time data sources would require substantial re-architecture.

Maintainability: The project's maintainability is high due to its modular design and adherence to best practices. Component replacement is straightforward, and dependency management and environment consistency are simplified. However, maintaining complex prompts and consistent agent behaviour, especially as models evolve, requires ongoing effort. Debugging agent interactions or external API issues can also be complex.

Limitations and Future Improvements

Beyond the core data limitations (source, age, brand coverage), the system is limited by its focus on text analysis (ignoring images/video) and the potential lack of nuance in understanding sentiment or complex interactions within the RAG analysis. The Competitor Finder's reliance solely on Exa Search also limits the breadth and depth of competitor discovery.

Potential future improvements include:

- **Data Source Expansion:** Integrating live data streams (e.g., Twitter API v2), other social platforms (Reddit, LinkedIn), news articles, and review sites.
- **Enhanced Competitor Finding:** Supplementing or replacing Exa Search with other data sources (e.g., industry databases, knowledge graphs) or employing different discovery techniques.
- **Richer Analysis Capabilities:** Adding features like sentiment trend analysis, topic modeling of support issues, automated SWOT analysis generation, and direct competitor comparison dashboards.
- **Robust Evaluation:** Implementing comprehensive quantitative evaluation for retrieval and generation quality.
- **Improved User Interface:** Developing more interactive visualizations and data exploration features within the Streamlit dashboard.

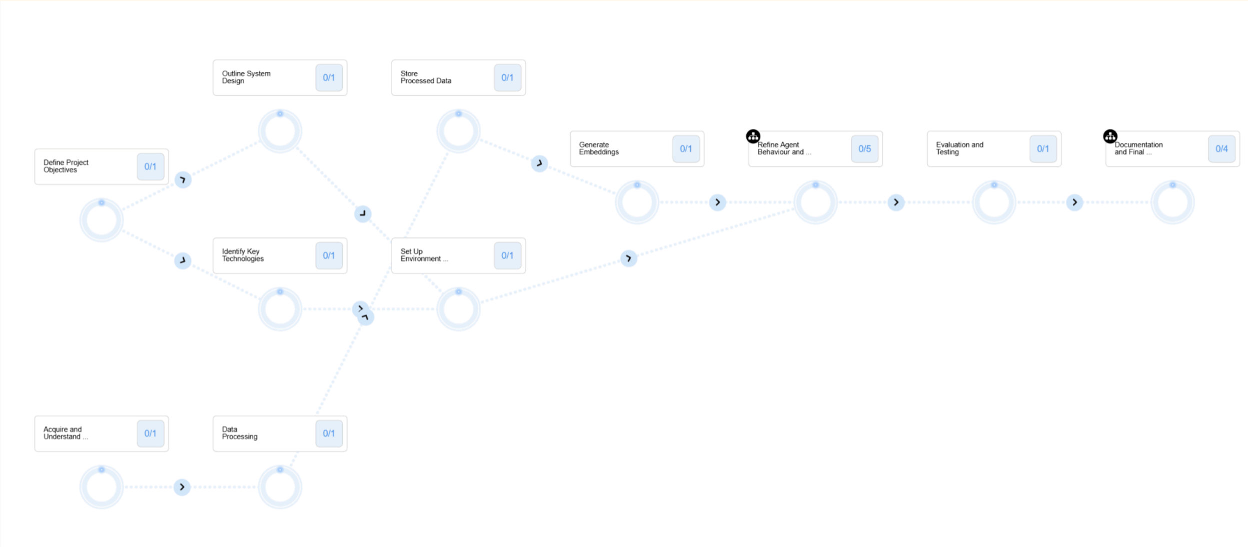
Conclusion

This project successfully demonstrated the design and implementation of an AI-Powered Competitor Intelligence Tool. By employing a dual-agent architecture built with modern AI and data engineering techniques, the tool addresses the need for automated competitor identification and in-depth customer interaction analysis. The Customer Interaction Analysis Agent, leveraging a robust RAG pipeline with Gemini embeddings and PgVector on historical Twitter data, proved effective, achieving strong retrieval performance as validated by RAGAS evaluation metrics (0.80 precision, 0.94 recall). The Competitor Finder Agent, while functional in leveraging the Exa Search API via MCP, showed performance inconsistencies, highlighting challenges in relying solely on current search-based agentic approaches for this task.

The project showcased best practices in software engineering, including containerization with Docker, efficient package management with UV, and secure configuration. While limitations exist regarding data sources, data recency, and the reliability of the Competitor Finder, the modular design provides a solid foundation for future enhancements. This work contributes a practical example of applying agentic AI and RAG systems to the domain of competitive intelligence, offering valuable insights and identifying key areas for continued development in building more comprehensive and reliable AI analysis tools.

The full project code can be found at
https://github.com/prathamskk/ucl_rag_mcp_ai_competitor_intelligence_tool

Appendix



Fractal provided a structured framework for defining these key phases and associated milestones, serving as a central tool for organizing the project's lifecycle and tracking its progression.