Competitive Market Research and GenAI

Utilizing Agent based AI to augment competitive market research.

John P Goodman

As a product manager, one of my ongoing tasks is to keep an eye on what competitors are doing, which often means diving into competitive market research. When I was at Georgia Tech working on my master's thesis, I focused on using analytics and machine learning to evaluate which companies were the most competitive. The toughest part of that project wasn't the analysis—it was gathering the data. I spent a lot of time scraping websites for product and marketing information, both manually and programmatically.

In this post, I want to talk about how agent-based AI, or agentic AI, can help make competitive research more efficient. These AI agents can handle a lot of the data collection and analysis, making it easier for product managers to get the insights they need without all the manual legwork. Let's take a look at how this approach could change the way we conduct competitive research.

Several emerging AI agent frameworks are specifically designed to help automate tasks like competitive market research. Some notable options include LangChain's LangGraph, Microsoft's Autogen, Hugging Face's Transformers Agents, OpenAI's Function-Calling API, Haystack Agents by deepset, and LlamaIndex's agent capabilities. Each of these frameworks allows for the creation and management of autonomous, task-oriented agents capable of handling data collection, processing, and analysis—tasks that typically require significant time and manual effort.

Among these, LangChain's LangGraph stands out as a powerful tool for building and managing complex AI workflows. It operates on a graph-based architecture, allowing you to set up interconnected agents, or "nodes," that perform specialized functions. LangGraph can handle various data sources, enabling agents to collect and synthesize data from multiple channels. This setup is particularly valuable in helping you to analyze information in a cohesive way to produce actionable insights.

To demonstrate how LangGraph can support this analysis of large amounts of competitive data, we will walk through a simple example where, we will set up a LangChain agent that queries the Tavily search engine, gathers relevant results, and then summarizes the findings to help us quickly gain insights without having to sift through endless sources of competitive information manually.

Build the agent:

To set up our LangChain agent, we will need the necessary API access keys. To begin, you'll need to sign up with OpenAI to obtain an API key. We will be using OpenAI to perform our summarization tasks. You'll need to add credits to your OpenAI account to ensure the model can handle multiple requests and that your API requests don't fail; Note that $5-$10 bucks should be sufficient for what we want to do; I added $20 dollars to my account and that has gone quite a long way.

Next, sign up for access to the Tavily search engine. After you sign up, there should be a default API key. Make sure to save both the OpenAI and Tavily somewhere such as an environment file (.env) or in your Jupyter Notebook. With these two API keys, our agent will be able to collect, process, and summarize data for competitive insights.

Now, let's dive into building our LangChain agent by setting up the necessary imports and configuring the API keys. First, we'll import the essential components from LangChain, including modules for creating an agent and managing API requests.

Here's a quick look at the initial setup and imports you'll need:

Link to GitHub Gist

Notice that we are importing a package called “newspaper”. This is a fork of Newspaper3k that has been updated and is known as Newspaper4k. When we issue a query against Tavily, the response will be a URL and some summary text. To get the full text of the URL, we will implement a quick and dirty web scraping function using Newspaper4k.

Link to GitHub Gist

Next, we will need to define our LLM and a state class. For the LLM, we will be using OpenAI’s GPT-4o.

The state class will hold the question to be answered, the answer from the LLM and the context which will any search results from Tavily; this class will hold the state of our agent.

Note the types in the state class. The state class is of TypedDict, both the question and answer should be strings and the context should be an annotated list.

Link to GitHub Gist

The next function we need to define is the search function. We will be using the Tavily search API for this function. Note that we are passing in the agent state and from the agent state we can pull in out question then pass it into the Tavily search function. We then take the results and pass the URL into our get_news_article_text web function, then format the output and return it as a dictionary with the key being “context”.

Link to GitHub Gist

The final function we need to define is the summarization function. This is where we will create a prompt for the LLM telling it to answer our question and pass it the results from the web search. Just like our previous functions, we pass in the state where we can grab the context and the original question. From there we create a prompt where the question and context are passed in. We then pass the prompt into the LLM for processing the return the answer as a dictionary.

Link to GitHub Gist

Now that we have all our functions defined, let’s actually build out the graph. Like any any graph, we will have nodes (or vertices) and edges. To define our graph, we need to call StateGraph and pass in our state class (TavilyState). To define the nodes (vertices) we can use the “add_node” method, give the node a name and pass it the functions we created.

Next, we need to define the edges of our graph; we can do that by calling the “add_edge” function. We will connect the special START node to the TavilySearch node, then connect TavilySearch to TavilySummary then connect TavilySummary to the special END node.

Link to GitHub Gist

Once we've defined the graph, we can compile it and see what it looks like. For this example, it's not very interesting since it's a simple directed graph that moves from START to END.

To run our agent, we can use the "invoke" method and pass in a dictionary with a key of "question" and a value that is our research question. In this example, we are asking our agent to identify the primary product categories and product features for Databricks.

Below is the output from our query. As you can see, while this is a very simple agent, the results are fairly comprehensive and demonstrates the power of agent-based AI.

```
The primary product categories and features for Databricks include:

1. **Lakehouse Architecture**: Databricks is renowned for its lakehouse architec

2. **Core Components**:
   - **Workspace**: A centralized environment for team collaboration, accessible
   - **Notebooks**: A version of Jupyter notebooks designed for collaboration an
   - **Apache Spark**: The engine for parallel processing of large datasets, sui
   - **Delta Lake**: Enhances data lakes with ACID transactions, ensuring data r
   - **Scalability**: The platform scales horizontally to meet increasing data d

3. **Key Features**:
   - **Cross-team Collaboration**: Enables seamless work among engineers, analys
   - **Efficient Workflows**: Supports data cleaning, transformation, and machin
   - **Integrated Data Management**: Allows data ingestion from multiple sources
   - **Real-time Collaboration**: Shared notebooks and collaborative editing fea
   - **Cluster Management**: Manages computational resources for executing code.
   - **Job Scheduling**: Executes notebooks and scripts at specified intervals.
   - **Data Ingestion**: Supports data import from various sources like AWS S3,
   - **SQL and BI**: Databricks SQL for running analytic queries and generating

4. **Advanced Capabilities**:
   - **Machine Learning and AI**: Tools for ML modeling, tracking, and model ser
   - **Generative AI Solutions**: Supports large language models and generative
   - **Data Governance**: Unity Catalog for managing permissions and secure data
   - **Serverless and Classic Compute**: Offers serverless compute for ease of u

5. **Integration and Open Source**: Databricks maintains a strong commitment to

Overall, Databricks provides a comprehensive platform for data processing, analy
```

If we curious about the data that our agent collected (as you should be), we can look at the context of our result simply by printing out the “context” key of our results. This will give us an idea of the data that the agent pulled in as well as serve as a sanity check to make sure our agent isn’t making things up.

Conclusion:

Even though this was a straightforward example, it’s easy to imagine how an AI agent could be configured to handle tasks beyond simple web searches. For instance, agents could be set up to monitor competitor websites for changes, track pricing or feature updates, analyze social media sentiment, or even compile insights from various market reports. This example highlights just how powerful AI agents can be in helping product managers streamline competitive market research. By automating repetitive tasks and gathering deeper insights, these agents free up more time for strategy, analysis, and decision-making—making them an invaluable tool in today’s fast-paced market landscape.