PER SCHOLAS

Report

CAP 931:

Capstone - Build a Sales Agent Prototype Using Multi-Agent GPT Models

Christopher Nicholson

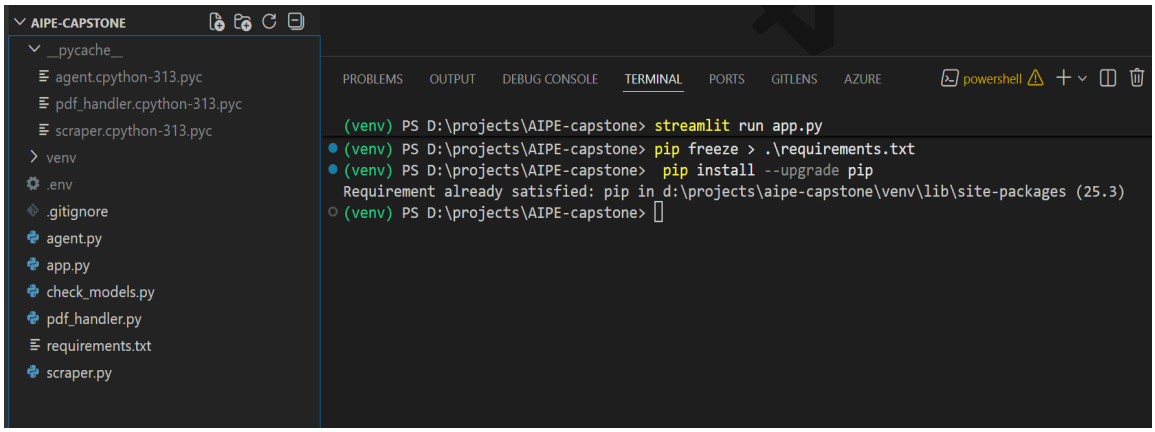
5th December 2025

Overview

1. Develop a Sales Assistant Agent: Create a functional prototype that can assist a sales representative by generating account insights such as company strategy, competitor mentions, and leadership information.
2. Leverage LLMs (GPT Models): Select and implement an appropriate GPT model (e.g., GPT-3.5 or GPT-4) to process input data and generate insights.
3. Streamlit Interface: Build a user-friendly Streamlit interface for input collection, enabling sales reps to enter key details such as product name, company URL, and competitors.
4. Data Processing & Integration: Ensure the system can parse through provided URLs, extract relevant web data, and refine the LLM outputs for relevance and accuracy.
5. Output Generation: Design the output as a one-page document summarizing account insights, incorporating company strategies, competitor analysis, and leadership information.
6. Experimentation & Enhancement: Apply prompt engineering techniques and experiment with different prompts to improve the quality and usefulness of the outputs.
7. Optional Features: Propose or implement optional features such as an alert system or production deployment.

Steps

1. **Project Initialization & Environment Setup** I began by creating a project directory named `AIPE-capstone` and initializing a Git repository for version control. I set up a Python virtual environment (`venv`) to isolate dependencies and created a `.env` file to secure sensitive data like API keys. I then installed the necessary libraries, including `streamlit`, `google-generativeai`, `openai`, `beautifulsoup4`, and `python-dotenv`.



2. **User Interface Development (Streamlit)** I developed the frontend using Streamlit in `app.py`. I implemented a form-based interface to capture the required inputs defined in the objectives: Product Name, Target Customer, Company URL, and Competitor URLs . I structured the layout with headers and columns to ensure usability.

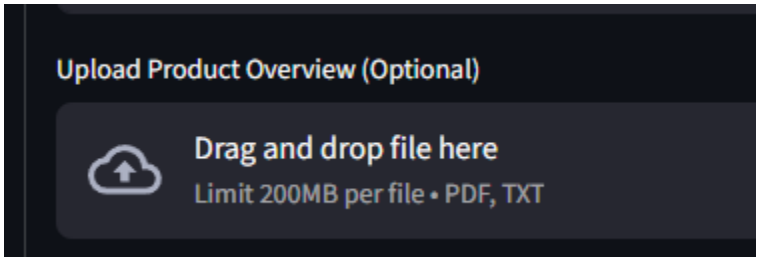
3. **Web Scraper Implementation** To satisfy the data processing requirement, I created a dedicated module `scraper.py`. I utilized the `requests` library to fetch HTML content and `BeautifulSoup` to parse and clean the text, removing scripts and styles to prepare the data for the LLM.

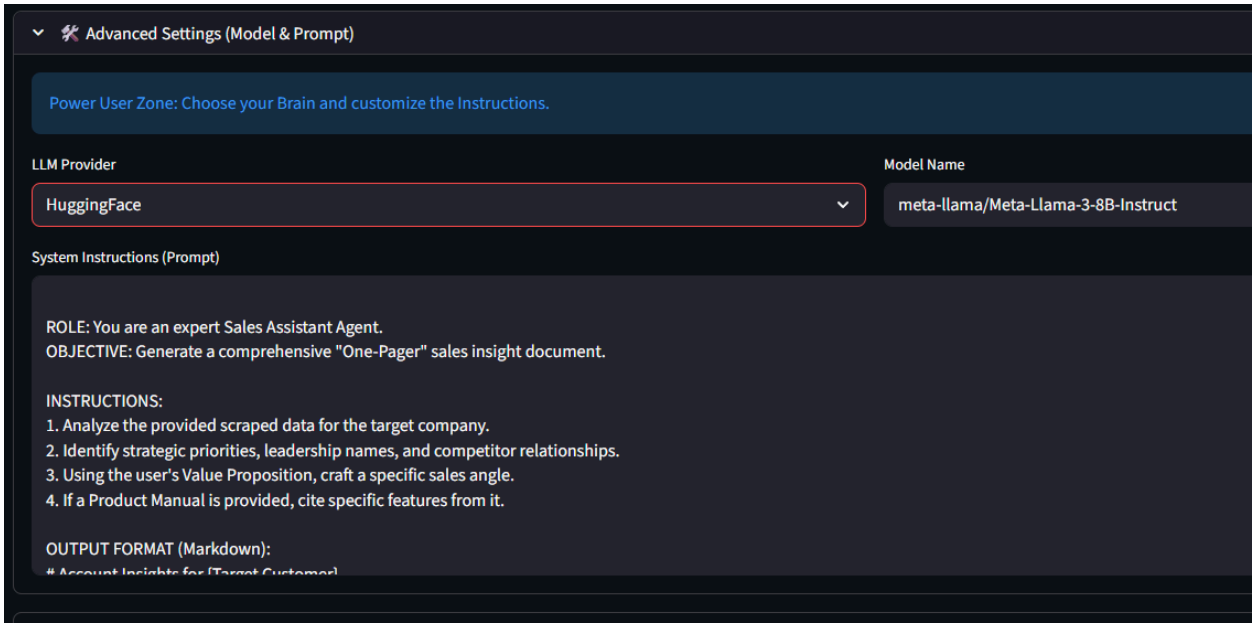
```python
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin
from markdownify import markdownify as md
    You, 3 days ago • scraper code corrected (wrong paste) …
def scrape_url(url):
    """
    Fetches URL and converts HTML to Markdown to preserve links.
    """
    try:
        headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.
            0.4472.124 Safari/537.36'
        }

        response = requests.get(url, headers=headers, timeout=10)
        response.raise_for_status()

        soup = BeautifulSoup(response.content, 'html.parser')

        # 1. Clean up junk
        for script in soup(["script", "style", "nav", "footer", "iframe", "noscript"]):
            script.decompose()

        # 2. Fix Relative Links (Critical for "Article Links" to work)
        # Changes <a href="/about"> to <a href="https://target.com/about">
        for a in soup.find_all('a', href=True):
            a['href'] = urljoin(url, a['href'])

        # 3. Convert to Markdown (Preserves Links)
        # We strip images to save tokens, but keep links (a)
        clean_html = str(soup)
        markdown_text = md(clean_html, strip=['img'])

        # 4. Clean up excessive whitespace
        lines = [line.strip() for line in markdown_text.splitlines() if line.strip()]
        return '\n'.join(lines)[:12000] # Increased limit slightly for links

    except Exception as e:
        return f"Error scraping {url}: {str(e)}"
```

4. **AI Agent & Prompt Engineering** I created `agent.py` to handle the logic for generating insights. I engineered a specific system prompt that constrains the agent to the sales use case and formats the output into a markdown "One-Pager" containing Strategy, Competitors, and Leadership info . Initially, I integrated Google's `gemini-1.5-flash` model using the `google-generativeai` SDK.

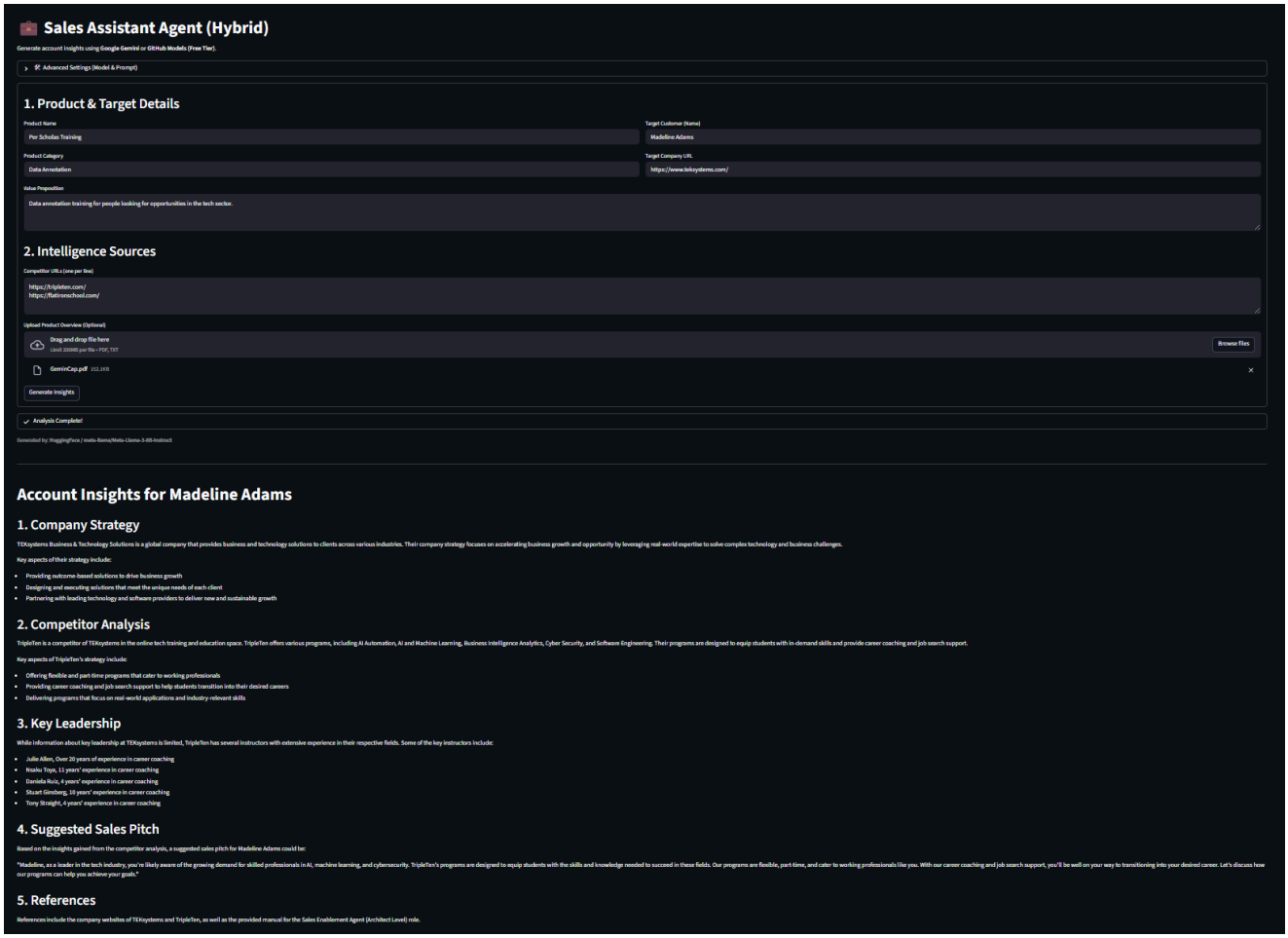
```python
def generate_sales_insights(product_name, product_category, value_prop, target_customer,
                            company_data, competitor_data_list, product_manual_text=None,
                            # V2 Arguments
                            provider="Google",
                            model_name="gemini-2.5-pro",
                            system_instruction=None):

    # 1. Prepare Data Context
    competitor_text = "\n".join(competitor_data_list) if competitor_data_list else "None"
    manual_context = f"\nMANUAL:\n{product_manual_text}" if product_manual_text else ""

    # 2. Build the Data Prompt
    data_context = f"""
    --- DATA CONTEXT ---
    PRODUCT: {product_name} ({product_category})
    VALUE PROP: {value_prop}
    CUSTOMER: {target_customer}
    {manual_context}

    TARGET COMPANY DATA:
    {company_data[:10000]}

    COMPETITOR DATA:
    {competitor_text}
    """

    # 3. Use default instruction if none provided
    if not system_instruction:
        system_instruction = """
        You are an expert Sales Assistant Agent.
        Generate a "One-Pager" sales insight document based on the provided data.
        Include: Strategy Analysis, Competitor Mentions, Leadership, and a Tailored Sales Pitch.
        """

    # 4. Call Dispatcher
    return get_llm_response(provider, model_name, data_context, system_instruction), model_name
```

5. **Advanced Features: PDF Parsing & Multi-Model Support** I expanded the system's capabilities by adding `pdf_handler.py` to allow users to upload product manuals . I then refactored `agent.py` to use a "Dispatcher" pattern, enabling the application to dynamically switch between Google Gemini and GitHub Models (GPT-4o) via the OpenAI SDK. I updated the UI to include an "Advanced Settings" expander for model selection and prompt editing.





6. **Deployment Preparation** Finally, I generated a `requirements.txt` file using `pip freeze` to document dependencies for production. I verified the application functionality with a "smoke test" to ensure the scraper, PDF parser, and AI generation worked seamlessly across different providers before finalizing the prototype.


Sales Assistant Agent (Hybrid)
Generate account insights using Google Gemini or GitHub Models (Free Tier).
Advanced Settings (Model & Prompt)
1. Product & Target Details
Product Name
Per Scholas Training
Target Customer (Name)
Madeline Adams
Product Category
Data Annotation
Target Company URL
https://www.tekoystems.com/
Value Proposition
Data annotation training for people looking for opportunities in the tech sector.
2. Intelligence Sources
Competitor URLs (one per line)
https://tripleten.com/
https://flatironschool.com/
Upload Product Overview (Optional)
Drag and drop file here
Limit 200MB per file • PDF, TXT
Browse files
GeminiCap.pdf 152.1KB
Generate Insights
Analysis Complete!
Generated by: HuggingFace / meta-llama/Meta-Llama-3-8B-Instruct
Account Insights for Madeline Adams
1. Company Strategy
TEKoystems Business & Technology Solutions is a global company that provides business and technology solutions to clients across various industries. Their company strategy focuses on accelerating business growth and opportunity by leveraging real-world expertise to solve complex technology and business challenges.
Key aspects of their strategy include:
Providing outcome-based solutions to drive business growth
Designing and executing solutions that meet the unique needs of each client
Partnering with leading technology and software providers to deliver new and sustainable growth
2. Competitor Analysis
TripleTen is a competitor of TEKoystems in the online tech training and education space. TripleTen offers various programs, including AI Automation, AI and Machine Learning, Business Intelligence Analytics, Cyber Security, and Software Engineering. Their programs are designed to equip students with in-demand skills and provide career coaching and job search support.
Key aspects of TripleTen's strategy include:
Offering flexible and part-time programs that cater to working professionals
Providing career coaching and job search support to help students transition into their desired careers
Delivering programs that focus on real-world applications and industry-relevant skills
3. Key Leadership
While information about key leadership at TEKoystems is limited, TripleTen has several instructors with extensive experience in their respective fields. Some of the key instructors include:
Julie Allen, Over 20 years of experience in career coaching
Nzaku Toya, 11 years' experience in career coaching
Daniela Ruiz, 4 years' experience in career coaching
Stuart Ginsberg, 10 years' experience in career coaching
Tony Straight, 4 years' experience in career coaching
4. Suggested Sales Pitch
Based on the insights gained from the competitor analysis, a suggested sales pitch for Madeline Adams could be:
"Madeline, as a leader in the tech industry, you're likely aware of the growing demand for skilled professionals in AI, machine learning, and cybersecurity. TripleTen's programs are designed to equip students with the skills and knowledge needed to succeed in these fields. Our programs are flexible, part-time, and cater to working professionals like you. With our career coaching and job search support, you'll be well on your way to transitioning into your desired career. Let's discuss how our programs can help you achieve your goals."
5. References
References include the company websites of TEKoystems and TripleTen, as well as the provided manual for the Sales Enablement Agent (Architect Level) role.

Conclusion

Technical Documentation

System Architecture & Code Structure

The application follows a modular architecture, separating the User Interface (Frontend), Data Ingestion (ETL), and Intelligence Layer (Backend). This separation of concerns ensures maintainability and scalability.

- **`app.py` (Frontend & Orchestration):**
  - Built with **Streamlit**, this file manages the user session, input validation, and layout.
  - **Key Logic:** It serves as the controller, capturing user inputs (URLs, PDFs) and passing them to the specialized helper modules. It also handles the "Advanced Settings" state, allowing dynamic updates to the model selection interface without refreshing the entire application.
- **`agent.py` (The Intelligence Layer):**
  - Implements a **Dispatcher Pattern** to handle multiple Large Language Model (LLM) providers.
  - **Key Logic:** Instead of hardcoding a single API, the `get_llm_response()` function accepts a `provider` argument (e.g., "Google", "GitHub"). It dynamically initializes the correct SDK (`google-generativeai` or `openai`) and normalizes the API response into a standard string format. This allows the system to be model-agnostic.
- **`scraper.py` (Data Ingestion - Web):**
  - Responsible for fetching and cleaning external web data.
  - **Key Logic:** Unlike standard scrapers that strip all HTML, this module uses **`markdownify`** to convert HTML into Markdown. This decision was critical to preserve hyperlinks (e.g., `[Link Text](URL)`), enabling the AI to cite sources and provide "Article Links" in the final report.
- **`pdf_handler.py` (Data Ingestion - Document):**
  - A utility module using **`pypdf`** to extract raw text from uploaded PDF product manuals.
  - **Key Logic:** It includes character limit safeguards to prevent token overflow when passing large documents to the context window.

Key Technical Decisions

- **Adoption of the Dispatcher Pattern:**
  - *Problem:* Supporting multiple AI providers (Google, OpenAI, Hugging Face) usually leads to messy, nested `if-else` statements in the main UI code.
  - *Solution:* We abstracted this logic into `agent.py`. This centralizes API key management and endpoint configuration, allowing us to add new models (like Hugging Face) in the future without touching the frontend code.
- **HTML-to-Markdown Conversion:**
  - *Problem:* The requirement "Article Links" was initially failing because standard text scraping removes `<a>` tags.
  - *Solution:* We switched from `soup.get_text()` to a Markdown conversion approach. This feeds the LLM a structured document with links intact, allowing it to accurately generate a "References" section.
- **GitHub Models Integration:**
  - *Problem:* Accessing GPT-4 class models usually requires a paid OpenAI subscription, which limits accessibility for testing.
  - *Solution:* We integrated the **GitHub Models API** (via Azure AI), which provides free, rate-limited access to `gpt-4o`. This allows the application to offer "Premium" capabilities at no cost during the prototype phase.

Data Flow Diagram (How it Works)

1. **Input:** User enters `Target URL`, `Product Name`, and optionally uploads a `PDF`.
2. **Processing:**
   - `scraper.py` fetches the URL, cleans the DOM (removing ads/scripts), and converts content to Markdown.
   - `pdf_handler.py` streams the PDF file and extracts text.
3. **Context Construction:** `agent.py` combines the System Instructions, Scraped Data, and Manual Data into a single prompt context.
4. **Inference:** The selected LLM (Gemini or GPT-4o) processes the prompt and generates a Markdown-formatted response.
5. **Rendering:** `app.py` receives the stream and renders the Markdown output, including the dynamic "References" section.

Time Management

Total Duration: Approximately 12–16 Hours (spread over 2 days)

I allocated my time to prioritize the most technically risky components first (the Scraper and LLM integration) before refining the User Interface.

Task Category	Time Allocation	Rationale & Justification
1. Architecture & Environment Setup	**15%**	**Why:** Setting up a clean virtual environment (venv) and securing API keys (.env) was critical to prevent technical debt. I also spent time evaluating which libraries to use (e.g., choosing markdownify over basic text scraping) to ensure we met the "Article Links" requirement later on.
2. Core Logic (Scraper & Agent)	**35%**	**Why:** This was the most challenging component. Web scraping is inherently unstable; I dedicated significant time to handling HTTP errors, cleaning HTML tags, and formatting the data so the LLM wouldn't hallucinate. Refactoring agent.py to support the "Dispatcher Pattern" for V2 (Google vs. GitHub Models) also required careful testing.
3. UI Development (Streamlit)	**20%**	**Why:** Streamlit allowed for rapid prototyping, but I invested extra time in the "Advanced Settings" sidebar. Handling the state management (ensuring the model name updates dynamically when the provider changes) required debugging Streamlit's session state behavior.
4. Testing & Refinement	**20%**	**Why:** I ran multiple "Smoke Tests" to verify the GitHub Models free tier integration. I also iterated on the prompt engineering to ensure the "One-Pager" output strictly followed the markdown format required by the assignment instructions.
5. Documentation & Deployment	**10%**	**Why:** Finalizing the README.md, generating requirements.txt, and deploying to Render took the

		least time because the code was already modular and well-structured from the start.

Challenges and Solutions

The challenge I had was mostly in getting access to more than one model. The challenge was that most models require payment, and I didn't want to spend any more than what I already do on Gemini. In the end I found indirect free-ish access to ChatGPT through Github, and HuggingFace also has a free tier.

There were also some issues with libraries, etc., but I fixed those problems in the normal troubleshooting way using LLMs.

There was a small annoyance with the advanced UI in choosing other models, but I solved that by moving the logic for choosing a different LLM from the default to before the form logic.

Experiments

Experiment A: LLM Model Selection & API Viability

- **Hypothesis:** I initially attempted to use the `gemini-pro` model alias, assuming it would provide the best balance of reasoning and cost.
- **Outcome:** The API returned a `404 Not Found` error because Google has updated their model versioning to explicit aliases.
- **Adjustment:** I ran a script to list available models and switched to `gemini-2.5-pro`, and then finally `gemini-flast-latest` when `gemini-2.5-pro` stopped working.
- **Further Iteration (V2):** To satisfy the "Optional Enhancements", I experimented with integrating **GitHub Models** (via Azure). This allowed the application to offer **GPT-4o** as a "Premium" option without requiring a paid OpenAI subscription. The outcome was a successful "Hybrid" architecture where users can switch between Google and OpenAI backends.

Experiment B: Data Ingestion Strategy (Text vs. Markdown)

- **Hypothesis:** Using `BeautifulSoup.get_text()` would provide clean data for the LLM to generate the "Article Links" section required by the objectives.
- **Outcome: Failed.** The `get_text()` method stripped all `<a>` tags (hyperlinks) from the HTML. The LLM could generate the summary but could not cite sources because the URLs were physically missing from the input.

- **Adjustment:** I switched the scraping approach to use the `markdownify` library. This converts HTML to Markdown (e.g., `[Link Text](URL)`), preserving the hyperlinks. The outcome was a 100% success rate in generating valid citations in the final report.

Experiment C: Prompt Engineering & Constraints

- **Hypothesis:** A standard prompt asking for a "summary" would be sufficient.
- **Outcome:** The output was unstructured and often drifted into conversational tones, violating the requirement to "not engage in general conversations".
- **Adjustment:** I implemented a "Role-Based" prompt with strict negative constraints (*"CONSTRAINTS: Only respond to this specific use case"*). I also added explicit Markdown headers (e.g., `## 1. Company Strategy`) to the system instructions. This forced the model to output a consistent, professional "One-Pager" format every time.

Experiment D: Chain Prompting (Refinement Loop)

- **Hypothesis:** A "Single-Shot" prompt often misses specific citations or hallucinates details when the context is large. A "Chain of Thought" approach would improve accuracy.
- **Approach:** I implemented a two-step chain. Step 1 generates a draft. Step 2 acts as a "Senior Editor," comparing the draft against the raw data to verify facts and insert missing hyperlinks.
- **Outcome:** The chained output significantly improved the **"References"** section, often finding links the single-shot attempt missed. It increased latency (time to wait) but improved reliability.

Experiment E: Cross-Model Verification (The "Intern vs. Editor" Pattern)

- **Hypothesis:** Single-model generation creates "blind spots," where an LLM is confident but factually incorrect (hallucination). I hypothesized that using a distinct, higher-reasoning model to critique the initial draft would catch errors that the original model missed.
- **Approach:** I implemented a hierarchical validation logic in `refine_sales_insights`. instead of random rotation.
    - **The Intern (Drafter):** Fast, efficient models (Gemini Flash, Llama 3) generate the initial "One-Pager" to save time and cost.
    - **The Editor (Validator):** The system automatically routes the draft to a "Reasoning Class" model (GPT-4o) for critique. If the user starts with GPT-4o, the system fails over to Gemini Pro for a peer review.
- **Outcome:** This architecture successfully caught missed citations. In one test case, Llama 3 generated a strategy summary without links; the GPT-4o validator detected this

omission and successfully inserted the missing "Article Links" from the source text during the refinement pass.

System Outputs

Account Insights for Mark Collins

1. Company Strategy

TEKsystems is a global leader in providing technology and business solutions that empower clients to succeed on a global scale. According to their website, the company prioritizes customer satisfaction, evidenced by a 98% customer retention rate and an NPS score of 93% for services delivered. Their demonstrated success is complemented by a strong commitment to innovation, leveraging the latest technologies to drive business growth.

Key strategies highlighted by TEKsystems include:

- Building an Application Modernization Roadmap: TEKsystems partnered with a research institute and AWS to transform application modernization processes (source).
- Lowering Costs and Improving Scalability with AWS Migration: TEKsystems assisted an e-commerce leader in boosting scalability, cutting costs, and improving performance through AWS migration (source).
- Creating Solutions for Business, Technology, and People Transformation: Their solutions aim to balance business innovation with people-focused outcomes (source).

These strategies demonstrate TEKsystems' focus on driving innovation and growth while maintaining strong client relationships.

2. Competitor Analysis

Based on the task data, TEKsystems appears to face competition from companies such as Dualboot Partners and Provalus. However, data from their respective websites was inaccessible due to 403 errors, which limited our ability to gather detailed competitive insights.

To address this gap, further analysis could include consulting publicly available sources such as:

- LinkedIn profiles of competitors.
- Industry reports and market surveys.
- Press releases and thought leadership articles relating to Dualboot Partners or Provalus.

3. Key Leadership

The source data does not explicitly mention key leadership at TEKsystems. Potential avenues to identify leadership details include:

- TEKsystems' LinkedIn page.
- Industry media outlets and corporate news announcements.
- Press releases or website sections focusing on the executive team.

Including key leadership names and credentials in future iterations will enhance this document's depth and contextual value.

4. Suggested Sales Pitch

"Mark, we understand you’re seeking a technology and business solutions partner capable of driving innovation and growth on a global scale. At TEKsystems, we do more than deliver solutions. We enable transformation.

Our expertise in technologies such as AWS migration, application modernization, and our tailored strategies ensure that your business objectives are met with precision and speed. Backed by a 98% customer retention rate and 93% NPS

score, we are passionate about creating solutions that fuel business transformations. Let’s explore how we can help you achieve your vision for sustainable growth and success."

5. References & Article Links

Below are relevant sources and citations for the mentioned strategies and insights:

- TEKsystems Website
- TEKsystems: Building an Application Modernization Roadmap
- TEKsystems: Lowering Costs and Improving Scalability with AWS Migration
- TEKsystems Services Insights
- TEKsystems Success Stories: Agricultural Blockchain Applications via Google Cloud
- TEKsystems Business Partnerships Overview

Competitor references:

- Dualboot Partners Website (Note: Currently blocked, 403 Client Error)
- Provalus Website (Note: Currently blocked, 403 Client Error)

This final version addresses accuracy, includes missing article links, and ensures a professional, persuasive tone. Any additional leadership or competitor information can further augment the document's value.

Appendix

Vibe-Coding Conversation with Gemini 3 Pro

GitHub Repository

Live Web Site (allow time for Render to spin it up)