n8n Integration Plan for Parley App AI Components

Introduction

This document outlines a comprehensive plan for integrating the Parley app's core AI components—the Professor Lock chatbot, the daily team picks (`teams.py`), player prop picks (`props.py`), and daily insights generation (`intelligent_professor_lock_insights.py`)—into an n8n workflow. A key focus of this integration is to implement a Retrieval Augmented Generation (RAG) system for Professor Lock, enabling it to access and leverage a rich knowledge base of company documents and gambling-related information. This integration aims to enhance the overall intelligence, efficiency, and responsiveness of the Parley app's AI capabilities.

1. n8n Capabilities for AI Orchestration and RAG System Integration

n8n is a powerful, open-source workflow automation tool that is well-suited for orchestrating AI workflows and integrating RAG systems. Its flexibility, extensive integrations, and ability to execute custom code make it an ideal choice for enhancing the Parley app's AI components.

1.1. AI Orchestration with n8n

n8n acts as a central command center for managing and coordinating various AI tools and services. It allows for the creation of complex, multi-step AI workflows by connecting different nodes, including those for interacting with Large Language Models (LLMs), external APIs, databases, and custom scripts [1].

Key capabilities for AI orchestration include:

- **LLM Integrations:** n8n provides native integrations with popular LLMs (e.g., OpenAI, Hugging Face models) through nodes like the "Basic LLM Chain" and "AI Agent" nodes.

This allows for sending prompts, receiving responses, and chaining multiple LLM calls within a single workflow [2].

- **Conditional Logic and Branching:** Workflows can incorporate conditional logic, enabling dynamic decision-making based on AI outputs or other data. This is crucial for creating adaptive AI agents that can respond differently based on context or user input.
- **Custom Code Execution:** n8n's "Code" node allows for executing custom JavaScript or Python code directly within a workflow. This is particularly useful for integrating existing scripts like `teams.py` and `props.py`, performing complex data transformations, or interacting with specialized APIs not covered by native integrations [3].
- **Scheduling and Triggers:** Workflows can be triggered by various events, including schedules (e.g., daily for pick generation), webhooks (for real-time chatbot interactions), or database changes. This ensures that AI processes run automatically when needed.
- **Error Handling and Logging:** n8n provides robust error handling mechanisms and logging capabilities, which are essential for monitoring the performance and reliability of AI workflows.

1.2. RAG System Integration with n8n

Retrieval Augmented Generation (RAG) systems enhance LLMs by providing them with access to external, up-to-date, and domain-specific information, thereby reducing hallucinations and improving the relevance and accuracy of responses. n8n is highly capable of building and integrating RAG systems [4].

The typical RAG workflow in n8n involves:

1. **Data Ingestion and Chunking:** Documents (e.g., company policies, gambling guides) are ingested, processed, and broken down into smaller, manageable chunks. n8n can handle various document types, including PDFs, text files, and web content [5].
2. **Embedding Generation:** Each text chunk is converted into a numerical vector (embedding) using an embedding model. These embeddings capture the semantic meaning of the text.

3. **Vector Database Storage:** The embeddings, along with their corresponding text chunks, are stored in a vector database. n8n integrates with popular vector databases like Pinecone, Supabase, Qdrant, and Weaviate [6].

4. **Query Embedding and Retrieval:** When a user asks a question, the query is also converted into an embedding. This query embedding is then used to search the vector database for the most semantically similar (relevant) text chunks.

5. **Context Augmentation:** The retrieved relevant text chunks are then passed to the LLM as part of the prompt, providing the LLM with specific context to generate a more informed and accurate response. This is where the "Retrieval Augmented" part of RAG comes into play [7].

n8n's "Vector Store Retriever" and "Vector Store Question Answer Tool" nodes facilitate this process, allowing for seamless integration of vector databases and LLMs to create powerful RAG-enabled applications [8].

2. n8n Workflow Architecture Design

To effectively integrate the Parley app's AI components, we propose a modular n8n workflow architecture consisting of distinct workflows for daily operations and the Professor Lock RAG system.

2.1. Daily Picks and Insights Generation Workflow

This workflow will be responsible for orchestrating the daily execution of `teams.py`, `props.py`, and `intelligent_professor_lock_insights.py`. It will be scheduled to run at a specific time each day (e.g., early morning before games start) to ensure fresh predictions and insights are available.

Workflow Steps:

1. **Scheduled Trigger:** A `Cron` node will initiate the workflow daily at a predefined time.

2. **Execute `teams.py`:** A `Code` node (Python) will execute the `teams.py` script. This script will fetch upcoming games, gather StatMuse and web search data, generate team picks

with reasoning, and store them in the Supabase `ai_predictions` table. The output of this node will be the generated team picks.

3. **Execute `props.py`:** Another `Code` node (Python) will execute the `props.py` script. Similar to `teams.py`, this script will fetch player props, perform research, generate player prop picks, and store them in the `ai_predictions` table. The output will be the generated player prop picks.

4. **Execute `intelligent_professor_lock_insights.py`:** A `Code` node (Python) will execute the `intelligent_professor_lock_insights.py` script. This script will likely analyze the generated picks and other data to create daily insights. The output will be the generated insights.

5. **Data Consolidation and Storage (Optional):** If needed, additional nodes can consolidate the outputs from `teams.py`, `props.py`, and `intelligent_professor_lock_insights.py` into a unified format or store them in other relevant database tables.

6. **Notification (Optional):** A `Notification` node (e.g., sending an email or a message to a Slack channel) can be added to alert administrators about the successful completion of the daily process or any errors encountered.

Benefits:

- **Automation:** Ensures daily picks and insights are generated automatically without manual intervention.
- **Centralized Control:** All daily AI processes are managed from a single n8n workflow.
- **Scalability:** Easily scale by adjusting resources allocated to the n8n instance.
- **Monitoring:** n8n's logging and execution history provide clear visibility into the process.

2.2. Professor Lock RAG System Workflow

This workflow will handle the ingestion of documents into the RAG system and serve as the retrieval mechanism for Professor Lock. It will be triggered on demand by Professor Lock's queries.

Workflow Steps (Ingestion - one-time or on-update):

1. **Manual Trigger / File Watcher:** For initial ingestion or when new documents are added/updated, this workflow can be triggered manually or by a `File Watcher` node monitoring a specific directory (e.g., for new company documents) or a `Webhook` if documents are pushed from an external system.
2. **Document Loader:** A `Read File` node or a custom `Code` node will load the documents (e.g., PDFs, Markdown files, text files). For different document types, appropriate parsing logic will be applied.
3. **Text Splitter:** A `Text Splitter` node will break down the loaded documents into smaller, manageable chunks. This is crucial for effective retrieval, as large documents can overwhelm the LLM's context window.
4. **Embedding Generation:** An `Embeddings` node (e.g., integrating with OpenAI's embedding API or a self-hosted embedding model) will generate vector embeddings for each text chunk.
5. **Vector Database Upsert:** A `Vector Store` node (e.g., Pinecone, Supabase, Qdrant) will store (upsert) the generated embeddings and their corresponding text chunks into the chosen vector database. Metadata (e.g., document source, creation date) can also be stored for filtering and relevance ranking.

Workflow Steps (Retrieval - on-demand by Professor Lock):

1. **Webhook Trigger:** A `Webhook` node will serve as the API endpoint that Professor Lock (or any other AI agent) calls when it needs to retrieve information from the knowledge base. The user's query will be passed as a parameter to this webhook.
2. **Query Embedding:** An `Embeddings` node will generate an embedding for the user's query.
3. **Vector Database Query:** A `Vector Store` node will query the vector database using the query embedding to find the most relevant document chunks. This node will return the top-k (e.g., top 5-10) most similar chunks.
4. **Context Assembly:** A `Code` node or `Merge` node will assemble the retrieved text chunks into a coherent context string, suitable for inclusion in the LLM's prompt.

5. **Respond to Webhook:** The assembled context will be returned to Professor Lock via the `Respond to Webhook` node.

Benefits:

- **Contextual Understanding:** Provides Professor Lock with access to a vast, up-to-date knowledge base, significantly improving the accuracy and relevance of its responses.
- **Reduced Hallucinations:** By grounding responses in factual information from the knowledge base, the RAG system minimizes the LLM's tendency to generate incorrect or fabricated information.
- **Scalable Knowledge:** Easily expand the knowledge base by adding new documents without retraining the LLM.
- **Maintainability:** Centralized management of document ingestion and retrieval processes.

3. RAG System Components and Data Sources

Building a robust RAG system for Professor Lock requires careful selection of components and a clear strategy for data ingestion. The goal is to provide Professor Lock with access to both internal company documents and external gambling-related information.

3.1. Vector Database Selection

The vector database is a critical component of the RAG system, responsible for efficiently storing and retrieving high-dimensional vector embeddings. Based on n8n's integrations and common industry practices, several suitable options exist:

- **Supabase (Postgres with `pgvector`):** Given that the Parley app already uses Supabase, leveraging its `pgvector` extension is a highly attractive option. This allows for storing embeddings directly within the existing database infrastructure, simplifying deployment and management. Supabase offers a managed service, reducing operational overhead. n8n has a dedicated Supabase Vector Store node, making integration straightforward [9].

- **Qdrant:** A popular open-source vector similarity search engine. Qdrant is known for its performance and rich filtering capabilities, which can be beneficial for more complex retrieval scenarios where specific metadata filtering is required (e.g., filtering gambling documents by sport or type). n8n has an integration for Qdrant [10].
- **Pinecone:** A managed vector database service optimized for large-scale, low-latency vector search. Pinecone is a good choice for production environments requiring high performance and scalability, though it comes with a cost [11].
- **Weaviate:** An open-source vector database that also offers a managed cloud service. Weaviate supports various data types and has a GraphQL API, providing flexibility in querying. n8n also integrates with Weaviate [12].

Recommendation: Given the existing Supabase infrastructure, **Supabase with `pgvector`** is the recommended choice for the vector database. It minimizes additional infrastructure complexity and leverages familiar tools, making development and maintenance more efficient.

3.2. Data Ingestion Methods

The process of getting documents into the vector database involves several steps: loading, parsing, chunking, embedding, and storing. n8n can automate much of this process.

3.2.1. Company Documents (Internal Knowledge Base)

These could include internal strategy documents, historical betting analysis reports, proprietary models, or FAQs about the app's features.

- **Sources:**
  - **Local Files/Cloud Storage:** Documents stored in a specific folder (e.g., on a server, Google Drive, Dropbox) can be monitored by n8n using `File Watcher` nodes or by periodically listing files and processing new/updated ones.
  - **Databases/CMS:** If internal documents are stored in a database or a Content Management System (CMS), n8n can connect directly to these sources to extract content.
- **Ingestion Process (n8n Workflow):**

1. **Trigger:** Manual trigger for initial bulk upload, or `File Watcher` / `Cron` for continuous updates.
2. **Load Document:** Use `Read File` node for local files, or relevant integration nodes (e.g., Google Drive, HTTP Request) for cloud storage/APIs.
3. **Parse Document:** For PDFs, use a library or service (potentially via a `Code` node) to extract text. For other formats (Markdown, plain text), direct reading is sufficient.
4. **Chunk Text:** Use n8n's `Text Splitter` node to break down large documents into smaller, semantically meaningful chunks. This is crucial for RAG, as it ensures that retrieved chunks are relevant and fit within the LLM's context window. Parameters like chunk size and overlap will need to be tuned.
5. **Generate Embeddings:** Pass the text chunks to an `Embeddings` node (e.g., OpenAI Embeddings, or a self-hosted model if privacy/cost is a concern) to convert them into vector representations.
6. **Store in Vector DB:** Use the `Supabase Vector Store` node to upsert the text chunks and their embeddings into the designated Supabase table.

3.2.2. Gambling-Related Documents (External Knowledge Base)

This category includes general sports betting guides, strategy articles, glossaries of terms, and public domain information that can enhance Professor Lock's general knowledge and provide authoritative answers.

- **Sources:**
  - **Web Scraping:** Use n8n's `HTTP Request` and `HTML Extract` nodes (or a dedicated web scraping service via `Code` node) to pull content from reputable sports betting education websites, forums, or regulatory bodies.
  - **Public APIs:** Some sports analytics or betting education platforms might offer APIs for accessing their content.
  - **Curated Datasets:** Manually curated datasets of gambling literature can be ingested.

- **Ingestion Process (n8n Workflow):** Similar to company documents, but with additional steps for web scraping and potentially more complex parsing of diverse web content. Regular scheduling will be important to keep this knowledge base up-to-date.

Data Quality and Maintenance:

- **Data Cleaning:** Implement steps in the n8n workflow to clean and preprocess text (e.g., remove boilerplate, ads, irrelevant sections) before chunking and embedding.
- **Metadata:** Attach relevant metadata to each chunk (e.g., source URL, topic, date published, author) to enable more precise retrieval and filtering by Professor Lock.
- **Update Strategy:** Define a strategy for updating the knowledge base (e.g., daily, weekly, or on-demand) to ensure Professor Lock always has access to the most current information.

4. Integration Methods for Professor Lock and Other AI Scripts

Integrating the existing Python scripts (`teams.py` , `props.py` , `intelligent_professor_lock_insights.py`) and Professor Lock with the n8n workflows and the RAG system is crucial for a cohesive AI ecosystem within the Parley app.

4.1. Integrating `teams.py` , `props.py` , and `intelligent_professor_lock_insights.py` with n8n

These Python scripts are currently responsible for generating daily picks and insights. Their integration with n8n will primarily involve triggering their execution and capturing their outputs.

- **Execution via n8n `Execute Command` or `Code` Node:**
    - **`Execute Command` Node:** For simpler scripts that can be run directly from the command line, the `Execute Command` node can be used. This node can execute a shell command (e.g., `python3 /path/to/script.py`).

    - `Code` **Node (Python):** For more complex interactions, or if the scripts require specific Python environments or libraries, the `Code` node (configured for Python) is ideal. This allows for direct execution of Python code within the n8n workflow, enabling more granular control over script execution, passing arguments, and capturing return values. This is the recommended approach for `teams.py`, `props.py`, and `intelligent_professor_lock_insights.py` due to their existing Python structure and potential for complex data handling.

- **Data Flow:**

    1. **Input:** The n8n workflow can pass necessary inputs to the Python scripts (e.g., current date, specific game IDs) as environment variables or command-line arguments, which the Python scripts can then parse.

    2. **Output:** The Python scripts should be modified to output their results (e.g., generated picks, insights) to standard output (stdout) in a structured format (e.g., JSON). The n8n `Code` node can then capture this output and pass it to subsequent nodes in the workflow for further processing (e.g., storing in the Supabase database, sending notifications).

- **Error Handling:** Implement robust error handling within both the Python scripts and the n8n workflow. If a script fails, the n8n workflow should catch the error, log it, and potentially send an alert to the administrators.

4.2. Professor Lock Leveraging the RAG System

Professor Lock, the chatbot, will be the primary consumer of the RAG system. The interaction will be initiated by user queries and will involve a real-time lookup in the knowledge base.

- **Chatbot-to-n8n Communication:**

    1. **Webhook Trigger:** As designed in Section 2.2, the n8n RAG retrieval workflow will expose a `Webhook` endpoint. When Professor Lock receives a user query that requires external knowledge, it will make an HTTP POST request to this n8n webhook.

2. **Query Payload:** The request payload will include the user's query and potentially other contextual information (e.g., user ID, conversation history snippet) that can aid in retrieval or personalization.

- **RAG Process within n8n:**

1. **Query Embedding:** The user's query received by the webhook will be embedded using the same embedding model used for the knowledge base documents.

2. **Vector Database Search:** The query embedding will be used to perform a similarity search in the Supabase `pgvector` database (or chosen vector DB) to retrieve the most relevant document chunks.

3. **Context Assembly:** The retrieved chunks will be assembled into a concise context string. This context will then be combined with the original user query and Professor Lock's system prompt.

4. **LLM Call:** This combined prompt (query + context + system prompt) will be sent to the LLM (Grok, or whichever LLM Professor Lock uses). The LLM will then generate a response grounded in the retrieved information.

5. **Response to Chatbot:** The LLM's generated response will be sent back to Professor Lock via the `Respond to Webhook` node. Professor Lock can then present this response to the user.

- **Enhancing Professor Lock's Prompt:**

  - **Tool Use:** Professor Lock's system prompt (in `claudeChatbotOrchestrator.ts`) should be updated to explicitly define the RAG system as a tool it can use. This involves instructing the LLM on *when* to call the RAG tool (e.g., when asked about specific company policies, historical betting strategies, or detailed gambling terms) and *how* to interpret the results returned by the RAG system.

  - **Contextual Awareness:** The prompt should also guide Professor Lock to understand that the information retrieved from the RAG system is authoritative and should be prioritized over its general knowledge for specific domain-related questions.

4.3. High-Level Architectural Diagram

To visualize the proposed integration, here is a high-level architectural diagram:

mermaid

Failed to render mermaid

Conclusion

Integrating the Parley app's AI components with n8n provides a robust, scalable, and flexible solution for orchestrating daily AI tasks and enhancing Professor Lock's capabilities with a RAG system. This approach centralizes AI workflow management, automates critical processes, and significantly improves the chatbot's ability to provide accurate, context-aware, and personalized responses by leveraging a rich, up-to-date knowledge base. The proposed architecture, utilizing Supabase with `pgvector` for the RAG system, offers a streamlined integration given the app's existing infrastructure. This strategic enhancement will undoubtedly elevate the user experience and the overall intelligence of the Parley app.

References

[1] n8n Blog: Your Guide to AI Orchestration: Best Practices and Tools: https://blog.n8n.io/ai-orchestration/
[2] n8n Docs: Basic LLM Chain node documentation: https://docs.n8n.io/integrations/builtin/cluster-nodes/root-nodes/n8n-nodes-langchain.chainllm/
[3] n8n Docs: Code node documentation:

https://docs.n8n.io/integrations/builtin/core/code/
[4] n8n Docs: RAG in n8n: https://docs.n8n.io/advanced-ai/rag-in-n8n/
[5] n8n Blog: How to build a RAG chatbot with n8n: https://blog.n8n.io/rag-chatbot/
[6] n8n Docs: What are vector databases?: https://docs.n8n.io/advanced-ai/examples/understand-vector-databases/
[7] n8n Docs: Vector Store Retriever integrations: https://n8n.io/integrations/vector-store-retriever/
[8] n8n Docs: Vector Store Question Answer Tool integrations: https://n8n.io/integrations/vector-store-tool/
[9] n8n Docs: Supabase Vector Store node documentation: https://docs.n8n.io/integrations/builtin/cluster-nodes/root-nodes/n8n-nodes-langchain.vectorstoresupabase/
[10] n8n Docs: Qdrant integrations: https://n8n.io/integrations/qdrant/
[11] Pinecone: https://www.pinecone.io/
[12] Weaviate: https://weaviate.io/