ChatGPT

Week 1 Hackathon Plan: AI Research Paper Assistant Kickoff

Overview and Objectives

In Week 1 of the GenAI Friday Hackathon, we will kick off the **AI-native R&D assistant** project. This project aims to automate the scanning of new AI research papers and extract insights that could improve an existing LLM-powered call center insights engine. According to the Quarter 1 plan, the first two weeks are about environment setup and foundational learning (Hugging Face course Chapters 1–2) **plus** some initial building [1]. Our goals for this week are:

- **Environment & Learning:** Set up a Python and Hugging Face development environment and complete Chapters 1 and 2 of the Hugging Face LLM course (covering transformer basics and capabilities).
- **Prototype Development:** Build a simple prototype that fetches a few recent research paper abstracts (e.g. from arXiv), summarizes them, and produces a **ranked list of suggestions** for how each paper could improve the call center LLM system.
- **Project Planning:** Define the mini-project scope (the R&D assistant) and outline a roadmap. Initialize a GitHub repository with a clear structure (code, README, etc.) and a list of future features.
- **Public Engagement:** Draft a LinkedIn post to announce the start of this Friday hackathon journey, outlining the project's goal and the learning path (Hugging Face course, etc.), setting context for our network [2].

By splitting time roughly 50/50 between **learning** and **building** [1], we'll reinforce new concepts from the course through hands-on practice. Below is the detailed plan and task breakdown for Week 1.

Part 1: Setup and Learning (Hugging Face Chapters 1–2)

Time allocation: ~2–3 hours (half of the session) for environment setup and learning.

- **Development Environment Setup:** Prepare a Python environment for GenAI work. This includes installing Python 3.x (if not already installed) and creating a dedicated virtual environment (using venv or Conda). We will install key libraries such as Hugging Face Transformers, Datasets, and PyTorch (or TensorFlow as needed) [3]. For example:

```
# Create and activate virtual environment
conda create -n ai-rnd-assistant python=3.10 -y
conda activate ai-rnd-assistant
```

```
# Install Hugging Face and other libraries
pip install transformers datasets torch chromadb langchain
```

The above ensures we have the transformers library for model pipelines, a dataset library, and even a vector DB (Chroma) and LangChain/LangGraph if we choose to use them. We'll also set up any necessary API keys or CLIs for modern GenAI tools. For instance, if planning to use **Gemini CLI**, we would install it (from its GitHub) and authenticate with a Google account for access to the Gemini 2.5 model [4] [5]. If using a cloud vector DB like Pinecone, we'd obtain an API key; for now, a local solution (Chroma) is fine.

- **Hugging Face LLM Course – Chapters 1 & 2:** Dive into the Hugging Face online course to refresh transformer basics and learn the Hugging Face ecosystem. Chapter 1 introduces the course and the transformer model concept, and Chapter 2 likely covers using pre-trained models and the Transformers pipeline for tasks [1]. During these chapters, we'll:
- Read about the Transformer architecture and the attention mechanism (as a refresher of fundamental concepts).
- Learn to use the Hugging Face `pipeline` for simple tasks. For example, a quick exercise from Chapter 1 might be using a pre-trained model to perform text generation or fill-mask. We will run a quick test like:

  ```python
  from transformers import pipeline
  fill_mask = pipeline("fill-mask", model="bert-base-uncased")
  result = fill_mask("Transformers are [MASK] for NLP.")
  print(result[0])
  ```

  This should output a prediction for the masked word (e.g., "amazing" or similar), demonstrating the pipeline functionality.
- Ensure we understand how to load models from the Hub and the basics of tokenization and inference. Chapter 2 may involve a simple text classification or generation task using a pipeline or the `AutoModel` classes.

- **Applied to Project Idea:** As we learn, we will keep in mind our project context (research-paper insights assistant). For example, when learning summarization in the course, note how it could apply to summarizing research abstracts. We will jot down any ideas on using transformers (like T5, BART, or GPT-style models) for summarization and how to evaluate relevance.

- **Plan the Quarter's Project:** Using the remaining time in this part, outline the scope of the "AI R&D assistant" project. The Quarter 1 guide suggests deciding on a mini-project in Week 1–2 (e.g. "summarize news articles") [6]. Our chosen project is **automating AI literature review** (scanning new papers for our LLM system). We will define the problem more concretely:

- *Scope:* Start with text summarization of abstracts and relevance ranking. Later, possibly expand to reading full papers or integrating with the production pipeline.
- *Relevance to course:* This project aligns well with NLP tasks like summarization and maybe classification (determining relevance), which matches the Hugging Face course content. It ensures we practice using transformer models for summarization and semantic similarity.

- *Success criteria:* By end of quarter, aim to have a working pipeline that, given a set of new research papers, can output a brief report of potential improvements for the call center LLM system.
- We will document these ideas in a **README** or planning doc in the repo.

- **LinkedIn Announcement Planning:** Per the Q1 plan, we will draft a LinkedIn post to announce the start of this Personal AI Hackathon [2] (see full draft at the end of this document). Key points to mention:

- Kicking off a weekly Friday-night AI project to build an AI-native R&D assistant.
- Goals for Q1 (completing the HF course and building a mini project).
- Excitement about learning LLM tech and applying it (inviting others to follow along).

By the end of this part, we should have our environment ready, foundational concepts reviewed, and a clear idea of what we're building. We'll then spend the next half focusing on the hands-on prototype.

Part 2: Prototype Development – Research Paper Scanner (Week 1 Mini-Project)

Time allocation: ~2–3 hours for coding and building a simple working prototype.

This prototype will be a **minimal end-to-end demo** of the R&D assistant's core functionality: retrieving a few recent AI research abstracts, summarizing them, and suggesting which are most relevant to the call center LLM system. We'll focus on a narrow slice to keep it achievable in one session.

2.1 Retrieving and Parsing Paper Abstracts: - We will start by obtaining 3–5 recent AI paper abstracts. For week 1, simplicity is key – we can use the arXiv API or even manually copy abstracts from arXiv's latest papers in a relevant category (e.g. **cs.CL (Computation and Language)** or **cs.AI**). A small Python script can use `requests` or an arXiv wrapper to fetch metadata. For example, arXiv's API allows querying by category and date:

```python
import requests
import xml.etree.ElementTree as ET

# Example: fetch latest 3 papers in cs.CL
url = "http://export.arxiv.org/api/query?
search_query=cat:cs.CL&max_results=3&sortBy=submittedDate&sortOrder=descending"
res = requests.get(url)
root = ET.fromstring(res.text)
entries = root.findall("{http://www.w3.org/2005/Atom}entry")
papers = []
for entry in entries:
    title = entry.find("{http://www.w3.org/2005/Atom}title").text
    abstract = entry.find("{http://www.w3.org/2005/Atom}summary").text
    papers.append((title, abstract))
```

This code (using arXiv's Atom feed) will yield a list of recent paper titles and abstracts. We will parse/clean the abstract text (the API returns summaries with newline characters and LaTeX, we might strip those for clarity). By the end of this step, we have a small **data sample** of 3–5 abstracts stored in a list or JSON file.

- *Alternate approach:* If arXiv API is slow to set up, as a one-off we might copy a few abstracts manually into a file for the prototype. The focus is on the pipeline rather than the data source in week 1. By future weeks, we'll automate this fully (possibly with a "Crawler Agent" to gather new papers daily, as others have done in multi-agent research workflows [7]).

2.2 Summarizing Abstracts with an LLM: - With the raw abstracts in hand, the next step is to **summarize each paper's key points and potential contributions**. We want summaries that highlight what the research is about and **how it might improve or relate to our call center LLM system**. - **Integration of a GenAI Tool:** For this summarization, we will leverage a modern GenAI tool: - *Option A: Gemini CLI:* Use Google's newly released **Gemini CLI** tool to summarize content via the Gemini 2.5 model in the terminal. For example, we can open the CLI, paste an abstract, and prompt Gemini: *"Summarize this research abstract in 3-4 sentences, focusing on any techniques that could improve a real-time call center LLM system."* The Gemini CLI, which offers powerful AI assistance in the terminal, should return a concise summary (Gemini is tuned for coding and content generation [5], and it provides a large context window for input [8] which is helpful for long abstracts). We will document the output for each paper. - *Option B: LangGraph Workflow:* Alternatively, we can set up a small **LangGraph** (a framework similar to LangChain) to automate this: define a graph where one node is a tool that inputs the abstract text into an LLM (like OpenAI GPT-4 or a local model) and outputs a summary. This could be done in code using a LangChain/LangGraph API. For instance, we might write a pseudo-code:

```
from langchain import OpenAI, PromptTemplate, LLMChain

prompt = PromptTemplate("Summarize the following paper and its relevance to a
call center LLM:\n{abstract}")
llm = OpenAI(model="gpt-3.5-turbo")  # or a local model
summarize_chain = LLMChain(llm=llm, prompt=prompt)

for title, abstract in papers:
    summary = summarize_chain.run({"abstract": abstract})
    print(f"**{title}** summary: {summary}\n")
```

This would yield a summary for each abstract. (*In practice, we could use Hugging Face Transformers pipeline for summarization as well, e.g. a BART or T5 model, but leveraging a powerful model via an agent or API might give more nuanced results.*) The key is we integrate an **LLM-driven summarizer** into our pipeline, either through a direct API call or an AI agent framework. - We will choose one of the above approaches to implement in Week 1. If time is limited or API access is an issue, we'll use a local summarization model via Hugging Face. For example:

```
from transformers import pipeline
summarizer = pipeline("summarization", model="facebook/bart-large-cnn")
for title, abstract in papers:
    summary = summarizer(abstract[:1024], max_length=100, min_length=30,
do_sample=False)
    print(f"{title} -> {summary[0]['summary_text']}")
```

This uses a CNN/DailyMail-tuned summarizer (BART) to condense the abstract. We truncate the input if it's long (1024 tokens) to fit model limits in this simple prototype. The result is a short paragraph highlighting each paper's gist.

- **Handling Domain Relevance:** We will prompt our summarizer (or manually ensure) to include hints of how the paper could relate to *improving an LLM system for call centers*. For instance, if a paper is about a new **fine-tuning method** or **dialogue model**, the summary should note that. (In future iterations, we might use a dedicated **Relevance Agent** that reads the summary and the system's description to score relevance – but for Week 1, a simple heuristic or manual judgement is fine.)

2.3 Creating a Ranked Suggestion List: - After summarizing, we will have a handful of paper summaries. The final step of the prototype is to output a **ranked list of suggestions** based on these papers. Essentially, answer: *Which of these new research findings are most promising for our call center LLM, and what changes do they suggest?* - **Ranking approach (Week 1):** Since we only have ~3–5 items, we can rank them manually or with a simple scoring: - If we have an embedding pipeline set up, we could vectorize each summary and compare it to an embedding of our target domain description (e.g., a short description of the call center insights engine task). Using a similarity metric (cosine similarity), we could order papers by relevance. For example:

```
from sentence_transformers import SentenceTransformer, util
model = SentenceTransformer('all-MiniLM-L6-v2')
system_desc = "Real-time call center conversation analysis and insight
generation."
desc_emb = model.encode(system_desc)
ranked = []
for title, summary in summarized_papers:
    emb = model.encode(summary)
    score = util.cos_sim(desc_emb, emb)
    ranked.append((score, title, summary))
ranked.sort(reverse=True, key=lambda x: x[0])
```

This would yield papers sorted by cosine similarity score to the system's description. (In practice, the MiniLM model provides a quick semantic similarity measure.) - Alternatively, we might simply **use the LLM** to evaluate relevance. For example, prompt an LLM: *"Here are summaries of 3 papers. Given the context of improving a call-center analytics LLM, rank these papers from most to least relevant and explain."* This could harness the reasoning ability of the model to do the ranking. For Week 1, this might be done manually (by reading the summaries and deciding) or with a quick prompt in Gemini CLI or ChatGPT if available. - **Suggestion list format:** We will compile the results into a neat list of 3–5 bullet points. Each item will include: - **Paper Title (Year)** – possibly with a short identifier. - A one-liner of the *suggested improvement* or key takeaway for our system, derived from that paper. - (Optionally) the relevance score or a tag like "High relevance" vs "Moderate relevance".

For example, a final output might look like: 1. **"Efficient Fine-Tuning for Dialogue Agents" (2025)** – Proposes a new fine-tuning method that could reduce our model's training time while improving response quality. *Suggested Action:* Try integrating their fine-tuning approach on our call center data to boost real-time summarization accuracy. 2. **"Real-Time Vector Search in LLM Pipelines" (2024)** – Introduces a vector database indexing strategy for faster knowledge retrieval. This could make our insight engine's FAQ matching module more responsive. *Suggested Action:* Prototype a vector DB (like FAISS or Chroma) for storing call transcripts and see if retrieval speed/accuracy improves. 3. **"XYZ" (2025)** – ... and so on.

(The above are illustrative; in the actual prototype, the content will come from the actual papers we summarized.)

By producing this ranked list, we achieve a tangible outcome for Week 1: a proof-of-concept where new research is distilled into concrete suggestions for the existing LLM system. This addresses the core goal of our AI R&D assistant in a very limited but functional way.

2.4 Integrating a Vector Database (tech preview): As an extra hands-on component, we set up a simple **vector database** (or embedding index) this week to demonstrate modern GenAI tooling. We installed Chroma or could use FAISS in memory. We will index our small set of paper embeddings into Chroma and run a similarity query:

```
import chromadb
client = chromadb.Client()
collection = client.create_collection("papers")
# Add papers with embeddings
for title, summary in summarized_papers:
    emb = model.encode(summary)
    collection.add(documents=[summary], embeddings=[emb.tolist()], ids=[title])
# Query by system description embedding
results = collection.query(query_embeddings=[desc_emb.tolist()], n_results=3)
print(results)
```

With only a few items, this is overkill, but it sets the stage for scaling up. In future weeks, a vector DB will allow us to store hundreds of paper embeddings and quickly find the most relevant ones as the literature grows.

Finally, we'll wrap up the prototype work by verifying that each piece (data fetch, summarization, ranking) is working in our notebook or script. The code and findings will be saved into our repository.

GitHub Repository Setup and Roadmap

We will create a new GitHub repository (perhaps named `ai-research-assistant` or similar) to host this project's code and documentation. Initial repository structure will be as follows:

```
ai-research-assistant/
├── data/               # Sample data such as saved abstracts, etc.
├── notebooks/          # (Optional) Jupyter notebooks for experimentation
├── src/                # Python source files for the project
│   ├── fetch_papers.py       # script to retrieve new papers (arXiv API
integration)
│   ├── summarize_papers.py   # script or module to summarize texts using LLM
│   ├── rank_suggestions.py   # script to rank papers and form suggestions
│   └── __init__.py           # (if needed for package structure)
├── README.md           # Project overview and instructions
└── requirements.txt    # List of dependencies for reproducibility
```

README Skeleton: We will draft a README.md with key sections: - **Project Overview:** Explanation of the problem (keeping up with AI research) and how our solution works (LLM-based summarization and suggestion). - **Installation:** How to set up the environment (e.g. clone repo, install requirements, set up API keys if needed). - **Usage:** How to run the prototype scripts or notebooks to see results. For example, usage instructions for fetching new papers and generating the suggestion list. - **Week 1 Prototype Details:** Describe what we built in Week 1 (e.g., summarizing a few papers). We might include an example output here. - **Roadmap / Future Work:** A bullet list of features and improvements to be implemented in upcoming weeks. For instance: - *Integrate full paper reading:* Use PDF parsing or APIs to analyze entire papers, not just abstracts. - *Automate continuous updates:* Schedule the system to fetch new papers weekly and maintain an updated suggestions list. - *Improve relevance scoring:* Implement a learning-to-rank model or more advanced semantic search tuned to our domain. - *UI or API:* Perhaps by end of quarter or next, develop a simple front-end or chat interface where users (developers) can query "What's new in AI for call centers?" and get answers from this assistant. - *Pipeline integration:* Eventually, allow the suggestions to directly feed into model fine-tuning experiments or A/B tests in the production LLM system.

We will add a note that this project is part of a learning journey, and we'll update the README each week as we progress (including links to any Hugging Face Spaces or demos in the future). By doing this in Week 1, we ensure good project hygiene and make it easier to share our code with others from the start.

LinkedIn Announcement Post (Draft)

(As our final task, we compose a LinkedIn post to mark the start of this journey, as encouraged by the Q1 plan [2] *. Below is a draft of the announcement post.)*

> **Excited to Kick Off My Personal Friday AI Hackathon!**
>
> Every Friday evening for the next few months, I'll be dedicating 4-6 hours to building my AI skills through a self-driven "Friday Night Hackathon." My goal for Quarter 1 is to **master Hugging Face's LLM course** and build a mini-project along the way.
>
> **Project Spotlight:** I'm creating an **AI-native R&D Assistant** – essentially a tool that reads the latest AI research papers and suggests how we can improve our own large language model systems (imagine having an AI intern that keeps you up-to-date!). The inspiration is to help a real-world LLM (like a call center insights engine) stay cutting-edge with minimal manual effort.

> **This Week's Plan:** Setting up my Python/ Transformers environment and completing Chapters 1-2 of the Hugging Face LLM course. I'll also prototype a small script to summarize a few new research papers and rank their relevance to our projects. It's a humble start, but I'm excited to learn by building!
>
> I'll be sharing weekly progress updates here – including challenges, breakthroughs, and tips I pick up. I'll also open-source the project on GitHub so others can follow along or even contribute.
>
> *Stay tuned* if you're curious about how AI can help us research... AI. Feel free to connect or comment if you're working on something similar! Let's learn together and push the boundaries of what AI can do.

This post outlines the context and invites our network to follow along, fulfilling the plan's suggestion to share our goals and progress publicly [2].

Sources Cited:

- The structure and timeline for Week 1–2 (environment setup and course kickoff) are based on the *Personal Friday Night AI Hackathon Plan* PDF [1] [2]. This guided us to split time between Hugging Face coursework and hands-on building.
- Google's **Gemini CLI** is referenced as a modern tool for leveraging the Gemini LLM in the terminal [4] [5]. Its open-source, CLI-based approach can assist with tasks like summarization.
- A relevant example of an AI agent pipeline for literature review by Z. Ahmad (2025) informed our vision of multi-agent stages (crawler, filter, reader) for future development [7], though our Week 1 prototype is a simpler subset of this idea.

---

[1] [2] [3] [6] Personal Friday Night AI Hackathon Plan (Next 3 Quarters).pdf
file://file-5iopN7LZspR3Qc6fuHvsLG

[4] [5] [8] Google announces Gemini CLI: your open-source AI agent
https://blog.google/technology/developers/introducing-gemini-cli-open-source-ai-agent/

[7] I Automated My Entire Research Workflow Using AI Agents — It's Like Having 5 Interns | by Zain Ahmad | Jul, 2025 | Artificial Intelligence in Plain English
https://ai.plainenglish.io/i-automated-my-entire-research-workflow-using-ai-agents-its-like-having-5-interns-f652374aa2a2?gi=2341597aa286