Tools and API Exploration

Assignment 1

Team 35

Esraa Nematalla **Osama Elkhuriby**

Tool / API Summary Table — Ollama

Feature	Explanation of Feature
Name of tool / API	Ollama
Link	https://ollama.com
Tool category (informal)	Foundation Model API / Local LLM runtime
Purpose	Provides access to pre-trained large language models (LLMs) for text generation, question answering, summarization, and reasoning. Supports integration into retrieval-augmented generation (RAG) pipelines with local execution for privacy-sensitive workflows.
Open-source / Proprietary	Proprietary (desktop and API-based models, local runtime option)
Authors / Organization	Ollama
Underlying programming language(s)	Python, Swift, Go, REST-based interface
Typical system role	Core LLM used for generating responses in RAG systems, chatbots, and domain-specific QA tasks. Supports both API calls and local model execution.
API or interaction style	Local API, REST endpoints, Python SDK
Output characteristics	Structured JSON responses containing generated text, tokens, and metadata; supports streaming for low-latency outputs.
Ease of use	High – easy local deployment, simple API calls, well-documented with example integrations.
Limitations and constraints	**Latency**: Local inference slower than cloud APIs (5–30+ sec depending on model size/hardware). **Hardware**: Requires sufficient RAM/GPU for running models locally (7B models need ~8GB RAM minimum). **Maintainability**: Manual model updates and resource management; version compatibility with client libraries needs monitoring. **Privacy**: Strong (fully local) but performance bottleneck is hardware capability, not API costs.
Relevance to final project	Will serve as the main Large Language Model (LLM) to generate answers from retrieved context in the RAG pipeline. Enables local execution for privacy and cost-free operation while producing coherent responses to AI programming questions.

Qdrant

Feature	Explanation
Name of tool / API	Qdrant
Link	https://qdrant.tech
Tool category (informal)	Vector Database / Retrieval Engine
Purpose	Stores and searches high-dimensional embeddings for semantic retrieval in RAG systems.
Open-source / Proprietary	Open-source (Apache 2.0) with managed cloud option
Authors / Organization	Qdrant Solutions
Underlying programming language(s)	Rust (core), Python/JS clients
Typical system role	Vector store for document chunks and query embeddings in RAG pipelines
API or interaction style	REST API, gRPC, Python SDK
Output characteristics	Top-k vectors with similarity scores and metadata filters
Ease of use	Very good documentation, simple setup (Docker/Cloud), strong filtering support
Limitations and constraints	**Maintainability**: Requires Docker or cloud deployment management; index rebuilding needed when changing embedding models. **Hardware**: Memory usage scales with number of vectors and dimensionality; ~1GB RAM per 1M vectors (approximate). **Latency**: Query speed depends on hardware and collection size; acceptable for small-scale projects (<100k vectors). **Privacy**: Fully local when self-hosted, ideal for sensitive data.
Relevance to final project	Stores embeddings of AI programming PDF chunks and enables fast semantic retrieval based on cosine similarity. Acts as the vector database layer in the RAG system, returning top-k most relevant document sections to ground the LLM's answers in source material

LangChain

Feature	Explanation
Name of tool / API	LangChain
Link	https://www.langchain.com
Tool category (informal)	LLM Orchestration & RAG Framework
Purpose	Provides building blocks to connect LLMs with external data sources, tools, memory, and reasoning chains for building RAG and agent-based systems.
Open-source / Proprietary	Open-source (MIT License)
Authors / Organization	LangChain Inc.
Underlying programming language(s)	Python, JavaScript
Typical system role	Orchestration layer in a GenAI pipeline (retrieval, prompting, chaining, memory, agents)
API or interaction style	Python / JS SDK, modular classes (Chains, Agents, Retrievers, Tools)
Output characteristics	Structured LLM responses, tool outputs, retrieved documents, agent action logs
Ease of use	Well-documented, large community, moderate learning curve due to many abstractions
Limitations and constraints	**Maintainability**: Frequent breaking changes between versions require careful dependency pinning and migration effort. **Latency**: Abstraction layer adds minor overhead compared to direct API calls. **Complexity**: Steep learning curve due to many abstractions; debugging can be challenging across chains. **Over-abstraction**: May be heavyweight for simple RAG use cases where direct integration is clearer.
Relevance to final project	Used to refactor the baseline RAG pipeline (Ollama + Qdrant) into a structured retrieval chain. Enables experimentation with prompt templates, chaining, and evaluation, demonstrating both foundational understanding and industry-standard orchestration.