**Project Citadel: Technology Alignment Analysis - Current vs. To-Be Architecture**

**Executive Summary**

Based on the analysis of the current Project Citadel implementation and the to-be architecture document, there are significant architectural shifts that require careful technology alignment. The most notable change is the transition from cloud-hosted LLMs (OpenAI) to on-premises models via **Ollama**, while maintaining **Crawl4AI** as a cornerstone component. This analysis identifies alignment gaps, required technology changes, and implementation considerations.

**1. Critical Architecture Alignment Analysis**

**1.1 Technology Stack Comparison**

Table

| **Component Category** | **Current Implementation** | **To-Be Architecture** | **Alignment Status** |
| --- | --- | --- | --- |
| **Web Crawling** | ✅ Crawl4AI 0.6.2 | ✅ Maintained (Core Component) | **ALIGNED** |
| **LLM Infrastructure** | ❌ OpenAI API (Cloud) | ✅ Ollama (On-Premises) | **MAJOR CHANGE** |
| **Vector Database** | ❌ ChromaDB | ✅ Qdrant | **REPLACEMENT REQUIRED** |
| **Backend Framework** | ❌ No FastAPI | ✅ FastAPI | **NEW ADDITION** |
| **Database** | ❌ File-based storage | ✅ PostgreSQL/Supabase | **MAJOR CHANGE** |
| **Orchestration** | ❌ Custom scripts | ✅ LangChain/LangGraph | **FRAMEWORK ADOPTION** |
| **UI Framework** | ✅ Streamlit | ❌ React/CopilotKit | **COMPLETE REPLACEMENT** |
| **Caching** | ❌ None | ✅ Redis | **NEW ADDITION** |

**1.2 Crawl4AI Integration Assessment**

mermaid

graph TB

subgraph "Current Crawl4AI Implementation"

CURRENT\_CRAWL[Crawl4AI 0.6.2]

CURRENT\_BROWSER[Playwright Browser]

CURRENT\_ASYNC[AsyncIO Framework]

CURRENT\_MEMORY[Memory Adaptive Dispatcher]

end

subgraph "To-Be Architecture Requirements"

TOBE\_CRAWL[Crawl4AI Core Engine]

TOBE\_FASTAPI[FastAPI Integration]

TOBE\_LANGCHAIN[LangChain Orchestration]

TOBE\_WORKERS[Crawler Worker Pods]

end

subgraph "Required Adaptations"

API\_WRAPPER[FastAPI Wrapper Layer]

WORKFLOW\_INT[LangChain Integration]

K8S\_DEPLOY[Kubernetes Deployment]

RESOURCE\_MGT[Resource Management]

end

CURRENT\_CRAWL -->|Preserve Core| TOBE\_CRAWL

CURRENT\_ASYNC -->|Maintain Pattern| TOBE\_FASTAPI

CURRENT\_MEMORY -->|Enhance for K8s| RESOURCE\_MGT

TOBE\_CRAWL --> API\_WRAPPER

TOBE\_FASTAPI --> WORKFLOW\_INT

TOBE\_LANGCHAIN --> K8S\_DEPLOY

classDef current fill:#ffcdd2

classDef tobe fill:#c8e6c9

classDef adaptation fill:#fff3e0

class CURRENT\_CRAWL,CURRENT\_BROWSER,CURRENT\_ASYNC,CURRENT\_MEMORY current

class TOBE\_CRAWL,TOBE\_FASTAPI,TOBE\_LANGCHAIN,TOBE\_WORKERS tobe

class API\_WRAPPER,WORKFLOW\_INT,K8S\_DEPLOY,RESOURCE\_MGT adaptation

**2. Major Technology Migration Requirements**

**2.1 LLM Infrastructure Migration: OpenAI → Ollama**

**Current Implementation Issues:**

python

# Current: OpenAI API dependency

from openai import OpenAI

client = OpenAI(api\_key=os.getenv("OPENAI\_API\_KEY"))

# Problem: Cloud dependency, cost per token, API rate limits

response = client.chat.completions.create(

model="gpt-4-turbo-mini",

messages=[{"role": "user", "content": prompt}]

)

**Required To-Be Implementation:**

python

# To-Be: Ollama integration with multiple approaches

import ollama

from langchain\_ollama import OllamaLLM

# Option 1: Direct Ollama client

ollama\_client = ollama.Client(host='http://localhost:11434')

# Option 2: LangChain integration

llm = OllamaLLM(

model="deepseek-r1:latest",

base\_url="http://ollama-service:11434"

)

# Option 3: FastAPI Gateway (Recommended)

async def ollama\_completion(prompt: str, model: str = "deepseek-r1:latest"):

response = await ollama\_client.chat(

model=model,

messages=[{"role": "user", "content": prompt}],

stream=True

)

return response

**Migration Complexity: HIGH**

* **API Compatibility**: Complete rewrite of LLM interaction layer
* **Model Management**: Local model deployment and versioning
* **Resource Requirements**: Significant compute and storage needs
* **Streaming Adaptation**: Different streaming protocols

**2.2 Vector Database Migration: ChromaDB → Qdrant**

**Current ChromaDB Implementation:**

python

# Current utils.py implementation

import chromadb

from chromadb.utils import embedding\_functions

client = chromadb.PersistentClient(path="./chroma\_db")

collection = client.get\_or\_create\_collection(

name="docs",

embedding\_function=embedding\_functions.SentenceTransformerEmbeddingFunction(

model\_name="all-MiniLM-L6-v2"

)

)

**Required Qdrant Implementation:**

python

# To-Be: Qdrant client implementation

from qdrant\_client import QdrantClient

from qdrant\_client.models import Distance, VectorParams, PointStruct

qdrant\_client = QdrantClient(host="qdrant-service", port=6333)

# Collection creation with proper configuration

qdrant\_client.create\_collection(

collection\_name="docs",

vectors\_config=VectorParams(

size=384, # all-MiniLM-L6-v2 dimensions

distance=Distance.COSINE

)

)

# Document insertion adaptation

points = [

PointStruct(

id=i,

vector=embedding,

payload={"text": doc, "metadata": meta}

)

for i, (doc, embedding, meta) in enumerate(document\_data)

]

qdrant\_client.upsert(collection\_name="docs", points=points)

**Migration Complexity: MEDIUM**

* **API Differences**: Different client libraries and methods
* **Data Migration**: Export/import of existing vector data
* **Configuration**: Different indexing and distance metrics
* **Performance Tuning**: New optimization parameters

**2.3 UI Framework Migration: Streamlit → React/CopilotKit**

**Current Streamlit Implementation:**

python

# Current streamlit\_app.py

import streamlit as st

from pydantic\_ai import Agent

st.title("Project Citadel RAG Interface")

user\_input = st.chat\_input("Ask about the documentation")

if user\_input:

with st.chat\_message("assistant"):

response = st.write\_stream(agent.run\_stream(user\_input))

**Required React/CopilotKit Implementation:**

typescript

// To-Be: React with CopilotKit

import { CopilotKit, CopilotSidebar } from "@copilotkit/react-core";

import { CopilotKitCSSProperties } from "@copilotkit/react-ui";

function CitadelRAGInterface() {

return (

<CopilotKit url="/api/copilotkit">

<div className="chat-interface">

<CopilotSidebar

instructions="You are an assistant for Project Citadel documentation"

defaultOpen={true}

labels={{

title: "Citadel Assistant",

initial: "Ask me about the documentation..."

}}

/>

<MainContent />

</div>

</CopilotKit>

);

}

**Migration Complexity: HIGH**

* **Complete Technology Shift**: Python → TypeScript/JavaScript
* **Framework Learning**: React patterns and CopilotKit APIs
* **State Management**: Different paradigms for conversation state
* **Integration Points**: New backend API contracts

**3. Required Technology Additions**

**3.1 FastAPI Backend Layer**

**Missing Component - Required Implementation:**

python

# New FastAPI application structure

from fastapi import FastAPI, BackgroundTasks, Depends

from langchain\_ollama import OllamaLLM

from qdrant\_client import QdrantClient

app = FastAPI(title="Project Citadel API", version="2.0.0")

# Dependency injection

def get\_ollama\_client():

return OllamaLLM(base\_url="http://ollama-service:11434")

def get\_qdrant\_client():

return QdrantClient(host="qdrant-service", port=6333)

@app.post("/crawl/url")

async def crawl\_url(

url: str,

background\_tasks: BackgroundTasks,

crawler\_type: str = "auto-detect"

):

# Integration with existing Crawl4AI components

pass

@app.post("/query")

async def query\_documents(

query: str,

llm: OllamaLLM = Depends(get\_ollama\_client),

vector\_db: QdrantClient = Depends(get\_qdrant\_client)

):

# RAG query implementation

pass

**3.2 LangChain/LangGraph Orchestration**

**Missing Component - Required Implementation:**

python

# LangChain workflow integration

from langchain.agents import AgentExecutor

from langchain.tools import Tool

from langgraph import StateGraph, END

# Crawl4AI tool wrapper

def crawl4ai\_tool(url: str) -> str:

"""Wrapper for existing Crawl4AI functionality"""

# Integrate existing crawler modules

from insert\_docs import smart\_chunk\_markdown

# ... existing implementation

return processed\_content

# LangGraph workflow definition

workflow = StateGraph(AgentState)

workflow.add\_node("crawl", crawl4ai\_tool)

workflow.add\_node("process", process\_content)

workflow.add\_node("store", store\_vectors)

workflow.add\_edge("crawl", "process")

workflow.add\_edge("process", "store")

workflow.add\_edge("store", END)

app = workflow.compile()

**3.3 Redis Caching Layer**

**Missing Component - Required Implementation:**

python

# Redis integration for caching

import redis.asyncio as redis

from fastapi\_cache import FastAPICache

from fastapi\_cache.backends.redis import RedisBackend

@app.on\_event("startup")

async def startup():

redis\_client = redis.from\_url(

"redis://redis-service:6379",

encoding="utf-8",

decode\_responses=True

)

FastAPICache.init(RedisBackend(redis\_client), prefix="citadel-cache")

# Cache integration with crawling results

@app.post("/crawl/url")

@cache(expire=3600) # 1 hour cache

async def cached\_crawl\_url(url: str):

# Cached crawling implementation

pass

**4. Critical Integration Points Analysis**

**4.1 Crawl4AI Integration Preservation**

**Strengths to Maintain:**

python

# Preserve these current patterns:

# 1. Memory Adaptive Dispatcher

dispatcher = MemoryAdaptiveDispatcher(

memory\_threshold\_percent=70.0,

check\_interval=1.0,

max\_session\_permit=max\_concurrent

)

# 2. Hierarchical Content Chunking

def smart\_chunk\_markdown(markdown: str, max\_len: int = 1000):

# This algorithm should be preserved

pass

# 3. Async Crawling Patterns

async def crawl\_recursive\_batch(urls, max\_depth=3):

# Current implementation is solid

pass

**Required Adaptations:**

python

# FastAPI wrapper for Crawl4AI

@app.post("/crawl/batch")

async def crawl\_batch\_endpoint(

crawl\_request: CrawlBatchRequest,

background\_tasks: BackgroundTasks

):

# Wrap existing crawl\_recursive\_batch functionality

result = await crawl\_recursive\_batch(

crawl\_request.urls,

crawl\_request.max\_depth

)

# Store results in Qdrant instead of ChromaDB

await store\_in\_qdrant(result)

return CrawlBatchResponse(status="completed", results=result)

**4.2 Model Resource Requirements**

**Ollama Model Resource Analysis:**

Table

| **Model** | **Size** | **RAM Required** | **Performance Impact** |
| --- | --- | --- | --- |
| deepseek-r1:latest | 4.7 GB | 8GB+ | Current implementation can't handle |
| deepseek-r1:32b | 19 GB | 32GB+ | Requires hardware upgrade |
| deepcoder:14b | 9.0 GB | 16GB+ | Moderate hardware requirements |
| mistral:latest | 4.1 GB | 8GB+ | Most compatible with current setup |

**Infrastructure Impact:**

yaml

# Required Kubernetes resource updates

apiVersion: apps/v1

kind: Deployment

metadata:

name: ollama-deepseek

spec:

template:

spec:

containers:

- name: ollama

image: ollama/ollama:latest

resources:

requests:

memory: "8Gi"

cpu: "2"

limits:

memory: "16Gi"

cpu: "4"

volumeMounts:

- name: model-storage

mountPath: /root/.ollama

volumes:

- name: model-storage

persistentVolumeClaim:

claimName: ollama-models-pvc

**5. Implementation Roadmap & Priority Matrix**

**5.1 Critical Path Dependencies**

mermaid

gantt

title Project Citadel Architecture Migration

dateFormat YYYY-MM-DD

section Infrastructure

Ollama Setup :crit, ollama, 2024-01-01, 2w

Qdrant Migration :crit, qdrant, 2024-01-08, 1w

PostgreSQL Setup :postgres, 2024-01-15, 1w

Redis Integration :redis, after postgres, 1w

section Backend Migration

FastAPI Framework :crit, fastapi, after ollama, 2w

LangChain Integration :langchain, after fastapi, 2w

Crawl4AI Adaptation :crawl4ai, after langchain, 1w

API Endpoints :apis, after crawl4ai, 1w

section Frontend Migration

React Setup :react, after apis, 2w

CopilotKit Integration :copilot, after react, 2w

UI Migration :ui, after copilot, 3w

section Testing & Deployment

Integration Testing :testing, after ui, 2w

K8s Deployment :k8s, after testing, 1w

Production Migration :prod, after k8s, 1w

**5.2 High-Priority Alignments**

Table

| **Priority** | **Component** | **Action Required** | **Timeline** | **Risk Level** |
| --- | --- | --- | --- | --- |
| **P0** | Ollama Integration | Complete LLM migration strategy | 2-3 weeks | HIGH |
| **P0** | Crawl4AI Preservation | Wrap existing functionality in FastAPI | 1-2 weeks | MEDIUM |
| **P1** | Qdrant Migration | Data migration and API adaptation | 1-2 weeks | MEDIUM |
| **P1** | FastAPI Backend | Core API framework implementation | 2-3 weeks | MEDIUM |
| **P2** | LangChain Integration | Workflow orchestration layer | 2-3 weeks | LOW |
| **P3** | React/CopilotKit UI | Complete frontend rewrite | 4-6 weeks | HIGH |

**6. Risk Assessment & Mitigation Strategies**

**6.1 Critical Risks**

**Risk 1: Ollama Performance vs. OpenAI**

* **Impact**: Slower response times, higher resource consumption
* **Mitigation**: Model selection optimization, hardware scaling, caching strategies
* **Fallback**: Hybrid approach with OpenAI API as backup

**Risk 2: Crawl4AI Integration Complexity**

* **Impact**: Loss of current crawling capabilities
* **Mitigation**: Phased migration, comprehensive testing, preserve core algorithms
* **Fallback**: Maintain current implementation in parallel during transition

**Risk 3: UI Technology Shift**

* **Impact**: Complete user interface rebuild required
* **Mitigation**: Incremental migration, user feedback loops, feature parity validation
* **Fallback**: Maintain Streamlit interface for internal use

**6.2 Technology Compatibility Matrix**

mermaid

graph LR

subgraph "High Compatibility"

CRAWL4AI[Crawl4AI 0.6.2] -->|Direct Integration| FASTAPI[FastAPI]

ASYNCIO[AsyncIO Patterns] -->|Preserved| LANGCHAIN[LangChain]

MEMORY[Memory Management] -->|Enhanced| K8S[Kubernetes]

end

subgraph "Medium Compatibility"

CHUNKING[Content Chunking] -->|API Adaptation| QDRANT[Qdrant]

METADATA[Metadata Extraction] -->|Schema Mapping| POSTGRES[PostgreSQL]

BATCH[Batch Processing] -->|Enhanced| REDIS[Redis Cache]

end

subgraph "Low Compatibility"

STREAMLIT[Streamlit UI] -.->|Complete Rewrite| REACT[React/CopilotKit]

CHROMADB[ChromaDB] -.->|Data Migration| QDRANT

OPENAI[OpenAI API] -.->|Protocol Change| OLLAMA[Ollama]

end

classDef high fill:#c8e6c9

classDef medium fill:#fff3e0

classDef low fill:#ffcdd2

class CRAWL4AI,ASYNCIO,MEMORY,FASTAPI,LANGCHAIN,K8S high

class CHUNKING,METADATA,BATCH,QDRANT,POSTGRES,REDIS medium

class STREAMLIT,CHROMADB,OPENAI,REACT,OLLAMA low

**7. Recommended Implementation Strategy**

**7.1 Phase 1: Infrastructure Foundation (Weeks 1-4)**

1. **Ollama Setup**: Deploy on-premises LLM infrastructure
2. **Qdrant Migration**: Vector database replacement
3. **FastAPI Framework**: Core API development
4. **Crawl4AI Wrapper**: Preserve existing crawling functionality

**7.2 Phase 2: Backend Integration (Weeks 5-8)**

1. **LangChain Integration**: Workflow orchestration
2. **API Development**: RESTful endpoints for all functionality
3. **Redis Caching**: Performance optimization layer
4. **PostgreSQL Integration**: Structured data storage

**7.3 Phase 3: Frontend Migration (Weeks 9-14)**

1. **React Foundation**: Base UI framework
2. **CopilotKit Integration**: AI-assisted interface
3. **Feature Parity**: Match current Streamlit functionality
4. **User Experience**: Enhanced interaction patterns

**7.4 Phase 4: Production Deployment (Weeks 15-16)**

1. **Kubernetes Deployment**: Container orchestration
2. **Monitoring Integration**: Observability stack
3. **Performance Tuning**: Optimization and scaling
4. **Migration Cutover**: Production transition

**8. Conclusion & Recommendations**

**Key Alignment Findings:**

* ✅ **Crawl4AI remains cornerstone**: Can be preserved with FastAPI wrapper
* ⚠️ **Major LLM infrastructure change**: Requires significant planning and resources
* ❌ **Complete UI technology shift**: High-risk, high-effort migration
* ✅ **Enhanced scalability**: Kubernetes deployment improves current limitations

**Critical Success Factors:**

1. **Preserve Crawl4AI Intelligence**: Maintain current crawling algorithms and memory management
2. **Plan Ollama Resource Requirements**: Ensure adequate compute resources for model hosting
3. **Phased Migration Approach**: Minimize disruption through incremental transitions
4. **Maintain Feature Parity**: Ensure no loss of current RAG capabilities

**Technology Alignment Score: 65%**

* High alignment for core crawling and processing capabilities
* Medium alignment for backend infrastructure
* Low alignment for LLM integration and UI technology

The migration to the to-be architecture represents a significant modernization effort that will improve scalability, maintainability, and on-premises AI capabilities while preserving the core intelligent crawling functionality that makes Project Citadel effective.