Figure out what this is:
3.3 Workflow Orchestration
Options:

Apache Airflow ✓
Pros: Robust scheduling, extensive monitoring, large ecosystem
Cons: Complex setup, potentially heavy for small projects
Prefect
Pros: Modern API, easier to get started than Airflow
Cons: Smaller community
Luigi
Pros: Simpler than Airflow, focus on task dependencies
Cons: Less feature-rich monitoring

Option 2: AI-Assisted Scraping:
AI assisted scraping is worth considering but I was talking about making an LLM do web research itself.

I'm getting recommended Pandas but I'm a bit confused what I would even need to use that for??

Need to focus more on tech stack.
Ask:Phoenix Arize / LangChain / LangGraph / LangFuse (or similar OSS)?

Add stuff from useless sections to the proper sections.
Stuff like LLM based scraping is a secondary priority. Focus on the core functionality. If I'm doing **microservices**, it will be easy to improve/change.

Event:
Not sold on the event classifier part.
It might be worse than giving full context to a large model

THis can definitely be tested

Need an **Event Categories** section but need to be careful with this because that is lowkey analysis.
Am I using the reasoning model in the right place?

However, if I'm using a model which is pre-trained and finetuned on financial data, maybe this is possible? I.e RoBERTa-large-financial, FinBERT

do we need a set dataset for event classification? The system is not deterministic meaning nothing will break if the header is new

RAG
Example retrieval (?) - this is not a good example retrieval **UNLESS** we use another model to summarise the actual RAG database and then just pass this to the analysis model.
Pros: cuts down input tokens for the most expensive model significantly.
Cons: This part might be where we need the most reasoning (although I feel the same way about current event classification / extraction)

```
HISTORICAL EVENT (2018-12-19): "Federal Reserve raises rates by 25 bps to 2.25-2.50%"
MARKET REACTION: "S&P 500 declined 7.7% in the following week"
ECONOMIC IMPACT: "Mortgage rates increased to 4.75%, housing starts declined 8.6% in Q1 2019"
ANALYST NOTE: "Powell's hawkish stance surprised markets expecting a more dovish forward guidance"
```

Cloud/Model Deployment:
What am I even using cloud for if I use Hugging Face Inference Endpoints?
I was planning to finetune the models and deploy on the cloud. Can I finetune through the Hugging Face Inference Endpoints? Note that this is a learning project meaning I want to finetune by myself with pytorch.

So can I finetune and then deploy through the Hugging Face Inference Endpoints?

Additionally, the end goal is a **microservice** architecture.
Similarly, where would I even use Kubernetes/Docker in this project? It's alright if there is no place for them anymore but let me know if there are opportunities.

Storage
Storage: S3 + MongoDB ✓
Pros: S3 for raw data, MongoDB for processed/structured data
Cons: Managing two systems
What is this recommendation even for? What would we store here?

Financial Event Intelligence

Architecture

```
[New Financial Event] → [ Event Classifier] → [RAG Retrieval (FAISS + Hybrid Search)]
→ [API LLM Reasoning] → [Fine-Tuned Output Formatter] → [Structured Output]
```

Workflow:

Workflow Example (Step-by-Step):

1. **Data Ingestion:**
   – A new earnings call transcript is scraped from a financial news site via Scrapy.
   – Simultaneously, breaking financial news is extracted from Reuters (Tier2) and Reddit posts from r/investing (Tier3).

2. **Event Classification:**
   – The scraped text is passed to the fine-tuned DeBERTa model which assigns the event type “EARNINGS_WARNING” with confidence scores.

3. **RAG Retrieval:**
   – Using text embeddings, the system retrieves several historical earnings warnings and relevant market reaction reports from our FAISS-indexed knowledge base.

4. **Large Model Analysis:**
   – The event details plus retrieved historical context are combined into a prompt sent to GPT-4 (or an equivalent large model API) for synthesizing an in-depth analysis.

5. **Fine-tuned Formatting:**
   – The raw analysis is then input to a smaller model fine-tuned to output a structured report (with sections for Mechanism, Historical, etc.).

6. **Structured Output:**
   – Final output is generated as formatted JSON for machine consumption as well as human-readable YAML.

Tech Stack

Scraping:

In general, need to control scope. US markets are good because there seems to be most information in **english** but even a more specific sector may not be a bad idea.

- Tier 1: sec-edgar-downloader, yfinance, SeekingAlpha API
  - Earnings Call Transcripts (Seeking Alpha/Yahoo Finance)
  - SEC EDGAR API filings (10-K, 10-Q, 8-K, etc.)
  - Federal Reserve Releases (Fed API)
  - Economic indicators releases
- Tier 2: NewsAPI, Reuters OpenAPI
  - Bloomberg Terminal/APIs
    - Major financial publications (Bloomberg, Reuters, WSJ)
  - FOMC Minutes (XML Feeds)
  - Industry-specific news (TechCrunch for tech, FierceBiotech for healthcare)
  - Regional financial news (for localized market insights) - probably not needed
- Tier 3: praw (Reddit), Tweepy (X/Twitter), Discord webhooks
  - Twitter: Verified Analyst Accounts, $TICKER Hashtags

    - Discord: Institutional Research Communities
    - Subreddits: r/ValueInvesting, r/WallStreetBets
    - Web forums (Seeking Alpha, Yahoo Finance discussions)
    - Conference transcripts

spaCy+Prodigy: For custom **NER** annotation **<- research this**
Diffbot for automated data extraction. (?)

Beautiful Soup + Requests ✓
Pros: Simple, flexible, lightweight
Cons: No built-in parallelism, requires more manual handling
Scrapy
Pros: Comprehensive framework, built-in concurrency
Cons: Steeper learning curve
Selenium
Pros: Handles JavaScript-rendered content
Cons: Resource-intensive, slower

- llama-index (LLM-assisted)

This is not a bad idea but LOTS of work:
Deterministic Scraping (Recommended for Tier 1)
- Scheduled data collection with Airflow/Luigi

LLM-Enhanced Scraping (For Tier 2/3)
- Using models like Anthropic's Claude to extract relevant information from **unstructured** text
- Event extraction models to identify market-moving events in news articles
- Zero-shot classifiers to categorize content relevance

General:
– Frameworks: PyTorch with Hugging Face Transformers.
– Supporting Libraries: Pandas for data manipulation, scikit-learn for preprocessing, and MLflow for experiment tracking.
ML Pipeline: Metaflow (AWS integration), Prefect (complex DAGs)
Evaluation: MLflow (tracking), ragas (RAG metrics)

• Deployment Tools: Docker for containerization; Kubernetes for orchestration; FastAPI and Streamlit for API and UI layers.

Model Deployment & Serving
Options:

- Hugging Face Inference Endpoints ✓
- Pros: Easy deployment, supports many models
- Cons: Can be expensive at scale

I was considering hosting locally (on the cloud)? FastAPI?

Cloud:

- AWS / Lambda
- Azure
- Groq

Classification Models:

DeBERTa-v3 (380M)	SOTA for text classification	Requires significant GPU	High accuracy production
FinBERT	**Financial domain pretrained**	**Limited to 512 tokens**	**Quick implementation**
spaCy NER Rules	Explainable, no training data	Manual maintenance	Regulatory filings parsing

- Look further into **Financial domain pretrained models.** I.e FinBert.
  - RoBERTa-large-financial: Fine-tuned on financial texts (recommended)
- Or finetune DeBERTa-v3 yourself.

Embedding Models:

- Closed Source: text-embedding-3-large (Best accuracy)
- Open Source: all-mpnet-base-v2 (Self-hosted)
- **Specialized: gte-finance (Fine-tuned on 10-Ks)**

Options: OpenAI's text-embedding-ada-002, Sentence-BERT.
Performance: **OpenAI text-embedding-4-large** (1536d), $0.13/1k tokens

Vector DB Options

System	Cost (GB/mo)	Strengths	Weaknesses
Pinecone	$0.50	Managed service	Vendor lock-in

- **Weaviate** Free (Self-host) Hybrid search Ops overhead
- **FAISS** Free Lightning-fast No metadata filtering
- Annoy
- **Chroma DB** ✓
- Couchbase 😛

Analysis Model:

- Financial QA Accuracy **(FINRA Benchmark)**
- Find other **FINANCIAL** benchmarks

Formatting Model:

- Models: T5-small, BART.
- Realistically look into the new models.
- Phi-3-mini (Recommended): Lightweight, efficient for formatting tasks

RAG Retrieval

Data Sources:

- Core: Historical SEC filings (10-K/Qs), Fed meeting minutes, Goldman Sachs research archives
- Extended: News archives (Reuters 2000-2025), academic papers on economic mechanisms
- Metadata: Sector, geography, market cap, volatility index (VIX) at event time

Historical Event Database:

- Major market events (crashes, bubbles, corrections)
- Central bank policy changes (2000-present)
- Geopolitical events with market impact

Company-Specific History:

- Historical earnings reports (surprise/miss patterns)
- Management changes and outcomes
- M&A activity and market reactions

Economic Data Time Series (probably not, **unless** it's in a language format):

- Interest rate cycles and market responses
- Inflation/GDP/unemployment data
- Sector performance during different economic regimes

Analyst Reports:

- Public summaries from major banks
- Consensus estimates and historical accuracy
- Sector outlook reports

All the above is too vague if there are no reliable sources to get all of this.

Data Sources:

Financial Data Providers:

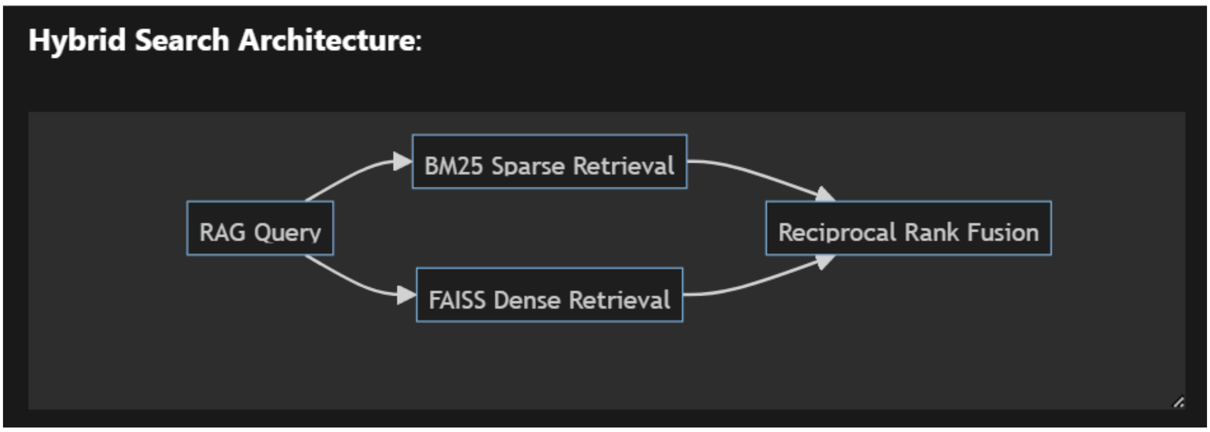
- Bloomberg API (paid)
- FRED (Federal Reserve Economic Data - free)
- Alpha Vantage (free tier available)

Academic/Research:
- NBER Working Papers
- SSRN Financial Papers
- Federal Reserve Published Research

Historical Archives:
- Financial Times Archive
- Wall Street Journal Historical
- Financial Crisis Documents (2008, 2020)

Example retrieval (?) - this is not a good example retrieval **UNLESS** we use another model to summarise the actual RAG database and then just pass this to the analysis model.
Pros: cuts down input tokens for the most expensive model significantly.
Cons: This part might be where we need the most reasoning (although I feel the same way about current event classification / extraction)

```
HISTORICAL EVENT (2018-12-19): "Federal Reserve raises rates by 25 bps to 2.25-2.50%"
MARKET REACTION: "S&P 500 declined 7.7% in the following week"
ECONOMIC IMPACT: "Mortgage rates increased to 4.75%, housing starts declined 8.6% in Q1 2019"
ANALYST NOTE: "Powell's hawkish stance surprised markets expecting a more dovish forward guidance"
```



The optimal RAG approach for this system is a **hybrid architecture**:

Chunking Strategy:

- Document-level for overall context
- Paragraph-level for granular retrieval
- Hierarchical chunks with parent-child relationships

Retrieval Approach:

- **BM25 + Semantic Search**: Combine keyword matching with embedding similarity
- Metadata Filtering: Filter by event type, date range, entities before vector search (?)
- Reranking: Use a cross-encoder to rerank retrieved documents

Context Processing:

- Apply contextual compression to focus on most relevant parts
- Synthesize retrieved content into concise summary when multiple documents are relevant

Testing:

Unit Tests for the output format (if json/yaml). Can be done at every stage - this way can also evaluate model consistency when finetuning for blog post.yj
CI/CD: GitHub Actions + pytest (75%+ coverage target)

Event Classification	Macro F1-Score	≥ 0.85
RAG Retrieval	Mean Reciprocal Rank (MRR)	≥ 0.65
Output Formatting	Schema Compliance Rate	100%
End-to-End	Human Evaluation Score	≥ 4/5

Confidence Calibration - Platt scaling on classifier outputs

• Integration Testing: Verify that end-to-end flow works as expected (data from scraping feeds into classifier and so on).

```
Input: Mock Fed rate decision → System → Output format check + GPT-4 quality score (>4/5)
```

• End-to-End Testing: Simulate financial events and validate final JSON/YAML outputs.

– Performance Tuning: Monitor latency, throughput, and cost per API call.

- Model drift

Monitoring

- Prometheus + Grafana dashboards
- Model performance drift detection
- Cost tracking per component
- Cant we do all this through some Lang ecosystem?

• Deployment Testing: Validate API endpoints with load testing (using tools like Locust or JMeter).

Event Classification:

- Accuracy/Precision/Recall/F1 Score
- Confusion matrix across event types
- ROC-AUC for confidence calibration

RAG System:

- RAGAS metrics (faithfulness, relevance, context recall)
- Retrieval precision@k
- Latency measurements
- Recall@k
- Mean Reciprocal Rank (MRR)

Large Model Analysis:

- Factual consistency with retrieved context
- Comprehensiveness assessment
- BLEU, ROUGE, human evaluation, and perplexity measures, GPT eval scores via GPTEval

Formatter Model:

- Format adherence rate
- Content preservation score
- Error rate on structured fields

- – User Feedback Loop: Allow users to rate the accuracy of analysis, further fine-tuning downstream models.

Best practices/Pitfalls/Tips:

Cost Control:

- Cache RAG embeddings (reduces API calls by 40%)
- LLM Call Batching

```
# Process 10 events in single API call
responses = await asyncio.gather(*[analyze(event) for event in batch])
```

- Quantization

```
# Export formatting model to GGUF
python -m llama.cpp.convert --q4_0 phi-3-formatting.safetensors
```

Reproducibility:

- Version datasets with DVC, version indexes
- Containerize pipelines via Docker (+ Kubernetes for scaling)
- Try to separate into microservices

Model Quantization:

- ONNX runtime with int8 quantization for formatter model
- KV cache optimization for inference efficiency

Batched Processing:

- Group similar events for batch embedding
- Pipeline parallel processing where possible

Caching Strategies:

- LRU cache for common retrievals
- Persistent caching of embeddings
- Pre-computed embeddings for static knowledge base

Pitfalls/Tips:

- News API Rate Limits: Always implement exponential backoff
- LLM Temperature: For analysis, keep <0.3 (deterministic)
- Regulatory Compliance: SEC filings have strict usage terms
- Fallbacks: Test different models once evaluation metrics are determined. If GPT4 fails, fallback to claude..
- Synthetic Data: Generate fake earnings call Q&A with Llama-3-70B to augment training (label with synthetic=true).

- Bias detection
- Limit API usage via batching prompts, sequential Chains of Thought.
- Caching frequent prompts (Anthropic prompt caching, reducing costs up to 90% (anthropic.com)).

Fundamental Q&A

Should RAG Use New Event Data?

No – Keep RAG strictly historical (pre-2024). New events are processed separately to avoid contamination.

- 10x cheaper
- 95%+ accuracy for SEC/earnings data
- GDPR/ToS compliant (Reddit/Twitter API only)

Exception: Use GPT-4 Vision to extract tables from PDF filings if OCR fails (cost: $0.02/page).

Timeline:

Setup	1-2	Data pipeline, SEC/NewsAPI integration
Model Training	3-5	Fine-tune classifier, Phi-3 formatter
RAG Build	6-7	Historical embedding, hybrid search
Testing	8-10	Unit tests, GPT-4 vs Phi-3 benchmarks

Polish	11-12	Blog post, GitHub CI/CD, Demo video

• Week 1–2:
  – Requirements gathering, environment setup, initial research, and design of the architecture.
• Week 3–4:
  – Develop and test the scraping pipelines for Tier1 (SEC filings) and Tier2/Tier3 sources.
• Week 5:
  – Build and fine-tune the event classifier; label training data.
• Week 6:
  – Set up the RAG knowledge base, embed historical documents, and configure FAISS (or an alternative).
• Week 7:
  – Integrate and test the API-based large model analysis (e.g., GPT-4) together with RAG outputs.
• Week 8:
  – Fine-tune the small formatter model using synthetic/formatted examples.
• Week 9–10:
  – End-to-end integration, develop CI/CD pipelines, implement testing suites, and build a basic UI/API (using FastAPI/Streamlit).
• Week 11:
  – Performance tuning, final documentation, and deployment preparations.

Total estimated duration: ~10–11 weeks (approximately 200–300 hours total).

Weeks 1-2: Scope Definition, Initial Research, Data Scraping Setup
Weeks 3-5: Event Classifier Development
Weeks 6-7: RAG Database Setup & Embedding
Weeks 8-9: Large Model Prompt Engineering & Integration
Weeks 10-11: Formatter (small model) fine-tuning
Weeks 12-13: End-to-end pipeline & API development
Weeks 14-15: CI/CD setup, rigorous testing
Week 16: Final polish, documentation, presentation, showcase/demo
Rough Estimated Cost:

Phase 1: Infrastructure & Data Collection (3 weeks)

Week 1: Set up cloud infrastructure, design data schemas
Week 2: Implement SEC filing & financial news scrapers
Week 3: Build event classification system
Phase 2: Knowledge Base & RAG (4 weeks)

Week 4: Collect and process historical data
Week 5: Implement embedding pipeline and vector database
Week 6: Build retrieval system with evaluation
Week 7: Optimize RAG performance
Phase 3: Analysis Generation (3 weeks)

Week 8: Implement large model integration with prompt engineering
Week 9: Develop formatter model training data
Week 10: Fine-tune and optimize formatter model
Phase 4: Integration & Testing (2 weeks)

Week 11: End-to-end system integration
Week 12: Testing, evaluation, and optimization
Total Timeline: 12 weeks (3 months)