Project InsightFlow: GenAI-Powered Transformation of Regulatory and News Feeds

by Dinis Cruz and ChatGPT Deep Research, 2025/05/03

Executive Summary

Company X's personalized news feed process can be transformed from a manual, labor-intensive workflow into a highly scalable, AI-assisted pipeline. We propose leveraging a GenAI-powered semantic knowledge graph approach – using the open-source stack developed by Dinis Cruz (OSBot, MGraph, MyFeeds.ai) – to automate the ingestion, tagging, and summarization of content for client-specific newsletters.

In this approach, source documents (e.g. financial regulations, compliance updates, industry news) are converted into **structured knowledge graphs** using large language models (LLMs), which capture key entities and their relationships. Simultaneously, a semantic profile graph is generated for each client or persona, representing their interests (e.g. a client's focus areas or a role-based profile like "CFO" or "Regulatory Analyst").

The system then **matches content graphs to persona graphs** to identify the most relevant updates and uses an LLM to generate tailored summaries for each reader. Crucially, every step produces **structured, traceable data** rather than opaque text, providing a clear provenance trail for why each article was selected.

The entire pipeline runs on a serverless architecture for elasticity and easy integration, with **human experts in the loop** to curate ontologies and approve final outputs.

This proposal outlines the problem with the current workflow, details the GenAI+Graph solution architecture, and provides an implementation plan for integrating these open-source tools into Company X's operations.

Problem Statement

Company X’s current workflow for producing personalized news feeds is heavily manual. Human analysts must gather relevant documents (financial reports, compliance notices, regulatory rulings, industry news articles), manually tag each with topics, and write summaries for different clients. While this human-driven approach can ensure quality, it suffers from several limitations:

- **Scalability and Efficiency:** As the volume of information grows, the manual process cannot keep up. Analysts can only process so many documents per day, limiting the number of clients or depth of personalization. Scaling up means hiring and training more analysts, which is costly and slow. There is a need for automation to handle **higher throughput** without sacrificing quality.
- **Consistency and Coverage:** Manual tagging is subjective; different analysts might categorize content differently, leading to inconsistent tagging or missed connections between related pieces of information. Important news could slip through cracks if an analyst isn’t familiar with a subtle topic link. For example, a regulation update mentioning “GraphQL security” might be missed for a tech-oriented client if not tagged with that keyword. An AI-driven semantic graph can automatically highlight such connections (e.g. article mentions *GraphQL* which matches a client interest in *APIs*), ensuring broader coverage.
- **Personalization Limits:** Today’s process likely uses broad categories to match articles to client profiles. Truly tailoring content (for example, focusing a summary on how a new regulation impacts *that specific client’s industry or past concerns*) is difficult to do manually at scale. The one-size-fits-all content risk is high. We need a way to deeply personalize each newsletter’s content and narrative angle to the client’s interests or role, which manual effort struggles to achieve beyond superficial filtering.
- **Lack of Explainable Rationale:** When a client asks “*Why am I seeing this article in my feed?*”, the answer currently depends on an individual analyst’s reasoning, which may not be formally documented. As the process scales, maintaining a clear rationale for each recommendation becomes challenging. The company would benefit from **provenance tracking** for recommendations – a transparent record of which client interest or profile data triggered the inclusion of each article. This is especially important in domains like finance and compliance, where clients will trust personalized content only if it’s credible and explainable.

In summary, the manual workflow is **time-consuming, difficult to scale, and lacks transparency**. It limits Company X’s ability to rapidly onboard more content sources or clients. The current system also makes it hard to ensure each client’s feed is both comprehensive and precisely tuned to their needs.

We need a solution that automates the heavy lifting of content processing and tagging, provides fine-grained personalization, and builds trust through explainability – all while keeping human experts in control of the curation.

Solution Overview

We propose a **Generative AI and Semantic Knowledge Graph** solution that automates the personalized feed generation process. The solution will ingest raw source materials and transform them through a multi-stage pipeline, mirroring the workflow proven in Dinis Cruz's Cyber Boardroom and MyFeeds.ai projects (which deliver personalized cybersecurity news feeds). The key idea is to represent both content and user profiles as graphs of semantic data, and use LLMs to bridge and summarize these graphs. Below, we outline the architecture and data flow:

1. Source Ingestion and Persistent Storage: All relevant source documents – including compliance documents, policy updates, regulatory rulings, news articles, RSS feeds, etc. – are ingested into a unified storage system. For example, an **ingestion script** (triggered automatically or by schedule) will fetch new items from RSS feeds or file repositories. Using the OSBot framework, this can be implemented as a serverless function (AWS Lambda) that gathers new content and saves it to an S3 bucket or database, with a timestamped folder structure for versioning. This persistent storage of raw data ensures we keep historical snapshots, enabling comparisons like "what changed since last week" (a capability already used to diff RSS feed snapshots in MyFeeds.ai). Storing raw content and intermediate results is important for traceability and re-processing if needed.

2. Transformation to Structured JSON (Content Graph Creation): Each ingested document is converted into a machine-readable **JSON representation of its key points**, using a large language model. In practice, an LLM (such as GPT-4 via API) is prompted to extract the *entities* (people, organizations, topics, etc.) and their *relationships* from the document's text. We use OpenAI's structured output mechanisms to have the LLM return a JSON object following a predefined schema (using Python classes or JSON schema). For example, for a news article, the LLM would identify entities like "Federal Reserve", "interest rates", "inflation" and relations such as *Federal Reserve -> raises -> interest rates*. This step essentially **encodes the who/what/how of the source material into a knowledge graph data structure**. Each document's JSON graph is stored (e.g. as `article_id.json`) in persistent storage.

- *Technology:* We will use **MGraph-DB** (open source library) to model and manipulate these graphs in code. MGraph allows treating JSON objects as nodes and edges in a graph, and provides utilities to merge graphs, query them, and even export visualizations. It's a "memory-first" graph database optimized for serverless use (works within Lambda memory and persists to JSON), which fits our architecture. After LLM extraction, we convert the output into an MGraph structure so we can easily perform set operations like union or diff, and generate visual diagrams for debugging if needed. By the end of this step, every source item yields a *semantic snapshot* of its content in graph form.

3. Ontology and Taxonomy Generation: In parallel with extracting content, the system can also **generate or apply a domain-specific ontology** – essentially the schema of entity types and relationships relevant to Company X's use case. In the initial phase, we might let the LLM freely choose relationships (as was done in the first MyFeeds MVP), which already produces useful results. However, to ensure consistency (e.g. always tag "GDPR" as a Regulation type, or link "Central Bank" to "Financial Authority" category), we will incorporate a predefined taxonomy of terms or allow the LLM to assist in building one. This is where humans provide oversight: domain experts at Company X can review the initial set of entity types/relations the LLM produced and **fine-tune the ontology** by defining standard categories. The LLM can then be guided to use this ontology in subsequent runs (for example, by providing it in the prompt or as few-shot examples). Over time, this ontology will evolve – with human-in-the-loop feedback ensuring it stays accurate. This approach mirrors the plan in MyFeeds.ai to move from free-form extraction to a "much more explicit set of entities and relationships" based on human feedback. A well-defined ontology will improve the quality of the graph and even reduce the need for an extra mapping step (since relevant connections could be found by simple graph traversal if entities share IDs or types).

4. Persona Profile Graphs (User Modeling): For each end-user or client who receives a newsletter, we construct a **persona semantic graph** representing their interests and context. In practice, Company X likely already segments clients by interests or industries (e.g. a client interested in "EU financial regulations" or a profile like "Tech Industry Investor"). We can take these profiles (in text form or as a list of interest keywords) and use an LLM to expand them into a richer graph of related concepts. For example, a persona description of *"CFO at a fintech company"* might yield a graph with nodes like "financial compliance", "blockchain technology", "start-up funding", and edges linking these to broader topics like "regulatory compliance -> fintech". This graph acts as a **semantic fingerprint of the user's interests**. Like the content graphs, persona graphs are stored as JSON. They can be partly predefined (and even edited by analysts to accurately reflect the client) and partly generated by the LLM to include hidden connections an expert might not list explicitly. (Notably, this can surface non-obvious interests; e.g., if a client follows *AI in finance*, the persona graph might include related terms like "algorithmic trading" even if the client didn't explicitly say so, because the LLM infers a relevant link). These persona graphs will be updated as client interests change or can remain static per client until edited.

5. Relevance Mapping (Matching Articles to Personas): With a content knowledge graph and a persona graph, the system's next job is to determine **which content pieces are relevant to which persona** and why. This is accomplished by another LLM-driven step that compares the two graphs for overlaps. Essentially, we prompt the LLM with the persona graph and an article's graph and ask it to find intersections – answering the question: *"Which entities or topics in this article align with the persona's interests?"*. The output is a structured JSON listing the specific matches and possibly a relevance score. For example, the LLM might output that *Article X* mentions **“Basel III”** which appears in the persona graph for a banking client, and **“cybersecurity”** which also appears, giving the article an 8.5/10 relevance score for that client. This stage effectively **links the article to the persona via common nodes** (entities) and provides the rationale for selection. Articles with no meaningful overlap will be filtered out, while those with several connections bubble up as top recommendations for that persona. All these mapping results are stored, building a **provenance trail** that we can later use to justify each recommendation (“we included this news because it connects to 3 topics you care about, X, Y, and Z”). If a strong ontology is in place, some of this mapping could be done with direct graph algorithms (e.g. checking for intersecting nodes), but using an LLM allows flexibility with synonyms and context (the LLM can recognize that “data protection” in an article is related to “privacy compliance” in the persona graph even if wording differs).

6. Personalised Summary Generation: In the final step for each piece of relevant content, the system generates a **tailored summary or write-up** for inclusion in the newsletter, addressing the persona’s perspective. Here we leverage the LLM one more time: it is prompted with (a) the original article text or key excerpts, and (b) the list of persona-specific relevant points from the previous step. The prompt is crafted to instruct the LLM to *“Write a brief summary of this article for [Persona X], focusing on the aspects that matter to them.”* Because we also feed in the structured connections (e.g. *Article mentions GDPR and Persona is interested in data privacy*), the LLM can explicitly tailor the summary to highlight those aspects. The output can be a short paragraph or a set of bullet points ready to go into the newsletter. We will enforce a structured format here as well – for example, the LLM could output a JSON with fields like `headline`, `summary_paragraph`, `key_points`, which we then render into final text. Having a structured output ensures the summary stays on task and we can easily concatenate multiple summaries consistently. **Hallucinations are minimized** because the LLM isn’t asked an open-ended question; it’s grounded in the article’s content and the known relevant facts (so it won’t introduce outside information). This step was demonstrated in MyFeeds.ai’s pipeline, where the LLM produced a JSON object for the personalised post, which was then converted to HTML/Markdown for publishing.

7. Assembly and Publishing: Once summaries are generated, the system compiles the personalized newsletter. This could be an automated email or a page on a portal. In Dinis Cruz’s implementation, the content was ultimately published to a Ghost CMS site via its API. For Company X, we might integrate with whatever delivery mechanism is in place (e.g. an email templating system or a web dashboard). The output for each client persona is essentially a curated list of summaries (with links to full sources) that the pipeline has determined to be most relevant for that period. Every piece of content in the newsletter can carry a tooltip or link to the **explanation (provenance)** of why it was included, drawn from the mapping step data. For internal use, each summary can be accompanied by the debug info: the underlying graphs and matching nodes, so editors can quickly verify the reasoning if needed.

Technical Architecture: The entire workflow above is orchestrated in a **serverless, modular pipeline.** We will use **OSBot-FastAPI** to set up the logic as a set of cloud functions (each major step can be an API endpoint or Lambda function). OSBot provides a scaffold to define these flows and deploy easily, with utilities for common tasks. For example, there will be endpoints like `/ingest-feed`, `/extract-entities`, `/build-persona`, `/match-graphs`,

`/generate-summary` corresponding to the stages. These can be composed into a sequence (potentially managed by OSBot-Utils' Flow system, which is inspired by Prefect's workflow engine). Because it's serverless, each step can scale independently – if a surge of new documents arrives, multiple Lambdas can run extraction in parallel. The *stateless* nature of Lambda is handled by our use of persistent storage (S3/DB) for passing data from one step to the next (one function writes the JSON output, the next reads it). We emphasize an **API-first design**, where each stage's output is available via API for monitoring or integration. This aligns with an "API-first architecture" for content and semantic operations, which enables flexible integration and reuse of the data. For instance, internal tools could query the knowledge graph API to find all clients interested in a given topic whenever a breaking news alert comes in.

Determinism and Explainability: By breaking the process into discrete, JSON-mediated steps, we ensure a degree of **deterministic behavior and transparency** in an otherwise complex AI pipeline. Every intermediate result (extracted entities, persona interests, mappings, scores) is saved and **auditable**, forming a complete narrative of how raw input became a newsletter output. This design means that given the same inputs, the system will follow the same decision path, and any variation (like an LLM wording difference) is confined and detectable (e.g. if an LLM ever produces malformed JSON, the system flags it). The use of structured outputs at each LLM step yields more consistent results and allows validation – if a field is missing, we know the AI deviated and we can correct it. In short, the solution treats LLMs not as a black box that magically goes from documents to newsletter, but as **co-pilots for well-defined tasks**, each constrained by schema. This yields a high level of explainability: we can **explain exactly why each article was selected** (because we have the list of matching entities as evidence), and we can trace how the summary was formed (since it's based on identified relevant points). This focus on provenance and determinism has been noted as a key to building trust in AI-driven content systems.

Role of Human Oversight: It's important to stress that this GenAI-powered system **augments** human analysts, rather than replacing them. Human expertise remains in the loop in several ways: (a) **Curation and Ontology tuning:** Experts will review the automatically extracted knowledge graphs and refine them. For example, if the LLM tags a concept incorrectly or uses an inconsistent name, analysts can correct the taxonomy. These corrections can be fed back to improve the prompts or even hard-coded as rules in MGraph (e.g. merge equivalent nodes, enforce that "UK" and "United Kingdom" are the same entity). The system is designed to learn from these adjustments over time, resulting in an evolving "expert-approved" knowledge graph of the domain. (b) **Approval of Outputs:** Especially during initial deployment, the content team might want to review the generated summaries before they go out to clients. The pipeline can output draft newsletters to an internal dashboard where a human reviewer quickly scans each item's summary and rationale. They can then approve, or in rare cases tweak the wording or remove an item. Because the summaries are backed by source content and graphs, an editor can trust that the factual content is from the source (minimizing fact-check needs, since we avoid hallucination). This review step ensures nothing sensitive or inappropriate is sent to clients and that the tone meets Company X's standards. (c) **Feedback loop:** Over time, feedback from readers or analysts (e.g. "client A found this article irrelevant") can be analyzed. If a mismatch occurs, it might indicate the persona profile needs updating or the mapping threshold for relevance should be adjusted. Humans will guide these strategic adjustments to continuously improve personalization accuracy. Ultimately, the system will handle the heavy lifting of processing and matching, **freeing the human analysts to focus on higher-level quality control and relationship management** with clients, rather than rote data processing.

By implementing this solution, Company X will gain a cutting-edge platform for personalized information delivery – one that is **AI-driven yet human-guided, scalable yet explainable**. The same workflow principles that have been applied in Cyber Boardroom and NewsFeeds.ai (for cybersecurity content) will be adapted to Company X's domain, demonstrating a strong continuity with proven techniques. Next, we outline a practical implementation plan, including phases and technical integration steps, to bring this solution to life.

Implementation Plan

Implementing this solution will be an iterative process. We will start with a **pilot project** focusing on a subset of content and a couple of persona profiles to validate the approach, then expand coverage and functionality. The following plan is proposed:

Phase 1: Setup Core Infrastructure and Pilot:

- **Serverless Pipeline Setup:** Using the OSBot-FastAPI framework, set up a basic **FastAPI service on AWS Lambda** (or Azure Functions, etc., depending on Company X's cloud preference) with endpoints corresponding to the key pipeline stages. We will initialize the project from OSBot's open-source template, which comes with default methods and a deployment setup that accelerates this process. This ensures from day one we have a working skeleton API where we can plug in our logic. The benefit of OSBot is that it simplifies the boilerplate of Lambda + API Gateway and provides an easy way to define *Flows* (sequences of tasks) that can run as one endpoint or step-by-step.
- **MGraph Integration:** Install and integrate the MGraph-DB library into the Lambda environment. We will create data models (Python classes or schemas) for our entities and relationships based on Dinis Cruz's examples. For the pilot, we may use the existing classes from the MyFeeds.ai project (which are generic, covering concepts like Entity, Relationship, etc.) to represent our knowledge graph. We will set up an S3 bucket (or a secure database) where all JSON outputs (feed data, article graph, persona graph, etc.) will be stored and versioned. The pipeline functions will use this for reading/writing intermediate state.

- **Pilot Data Selection:** Choose a limited but representative set of sources and one or two personas for the pilot. For example, we might start with *“Monthly Financial Regulation Updates”* as the content stream (perhaps using regulatory news from a known source) and two persona profiles: one for a **Financial Services Executive** and one for a **Compliance Officer**. These personas can be defined in text and then converted to graphs. Keeping the scope narrow initially allows us to develop and tune the system in a controlled setting.
- **LLM Connectivity:** Set up access to a suitable LLM. Initially, this could be OpenAI’s GPT-4 via API (which was used in the reference implementation). We will utilize the function calling / structured output feature to get JSON back directly. We will also implement an **LLM caching mechanism** (as done in MyFeeds.ai) to store LLM responses for given inputs. This avoids re-calling the model for the same article multiple times and ensures consistency (the same text will yield the same extracted entities, retrieved from cache). If needed due to data sensitivity, we can explore using a self-hosted LLM (like Llama 2 or an open model) – the architecture allows swapping the LLM service without fundamental changes, since we control the prompts and expected output format.
- **Prototype the Stages:** Implement each stage in a basic form:
- Write a function to fetch or receive a document and produce `feed_article.json` (this can be trivial if we start with already available text).
- Implement the **Entity Extraction** step with a prompt that asks the LLM to fill the JSON schema with entities/relations for a given text. Test it on a few sample inputs and adjust the prompt or schema as needed to capture the information we expect.
- Implement the **Persona Graph** generation with a prompt that takes a persona description and yields key interest entities (we can use the approach from MyFeeds, feeding the persona text into a system prompt that asks for a similar JSON structure of interests).
- Implement a simple **Matching** step: this could use an LLM prompt that includes both JSON graphs (article and persona) and asks for common elements. For the pilot, even a simpler algorithmic approach (like finding common words or categories) can be used first to validate the concept, then replaced with the LLM for nuance.
- Implement the **Summary Generation** by prompting the LLM with a template that includes the article title, maybe a short excerpt or the whole text (if short), and the list of matched interests, asking for a paragraph aimed at the persona.

Each of these functions will be available via the API and also callable as an orchestrated flow for a given new article. At this stage, we will run them manually in sequence (or write a small script to simulate the pipeline) to verify each piece works.

- **Verification and Tuning:** Have Company X’s content experts review the outputs of each stage. For example, take a sample news article and see if the extracted entity graph makes sense (Are important concepts captured? Is any extraneous or hallucinated relation present?). Adjust prompt instructions or the schema if needed. This is where we might start shaping the ontology – e.g. we notice the LLM output uses “org” vs “organization” inconsistently, so we enforce one. Or we see an entity type we don’t care about and decide to ignore it. This feedback is fed into either code (post-process the JSON to drop unwanted parts) or prompt (ask LLM to exclude certain things). We also tune the persona graphs: ensure the generated interests reflect what we expect for that role/persona. This step might involve editing the persona description or providing the LLM a few examples of what interests to output.
- **Success Criteria for Phase 1:** By the end of Phase 1, we should have a working pipeline that can take a new article and produce a draft personalized summary for our test persona(s), along with the evidence of why it was chosen. We will have demonstrated the end-to-end flow on a small scale. The technical team and stakeholders can evaluate the quality and usefulness. Key metrics to assess: Does the summary indeed focus on the persona’s interests? Are the reasons for selection clear and correct? We expect some iteration here, especially on prompt engineering, to hit an acceptable accuracy.

Phase 2: Scale Up Sources and Automation:

- **Expand Content Sources:** Once the approach is validated on a small scale, we will integrate more of Company X’s content streams. This might include connecting to live RSS feeds of relevant news sites, setting up scrapers or API calls for regulatory portals, or ingesting internal research documents. Each new source can reuse the ingestion mechanism we built (perhaps just adding new feed URLs to fetch). We will also utilize the **timeline diffing** technique from MyFeeds to handle feeds that update regularly – storing the last seen items and computing what’s new since last run. This ensures we process only new documents each cycle and know when items are removed or updated. MGraph’s ability to diff two sets (previous vs current snapshot) will be helpful here.
- **Add More Personas/Profiles:** We will work with Company X to define additional persona profiles (likely corresponding to different client segments or roles). For each, we either write a short description or provide a list of interest keywords. These are then run through the persona graph generator to yield a JSON profile. We’ll store these profiles in a database so that the pipeline can reference them when generating newsletters. If there are many clients each with slight variations, we might group them into a manageable number of persona archetypes to start with (to avoid having to run LLM steps separately for each individual if not necessary). Over time, we could even allow each client to have a truly custom profile, but initially focus on categories (e.g. by industry or job role).

- **Automation & Scheduling:** Set up automated triggers for the pipeline. For example, a CloudWatch timer (or equivalent) could trigger an ingest run every morning, which pulls new content and then kicks off processing. Because our pipeline is serverless and composed of independent steps, we might use a lightweight orchestration: for instance, a Lambda function that on schedule lists new items, and for each item calls the sequence: extract -> match -> summarize -> store result. We can also batch this per persona or per item as needed. Another approach is to use an orchestration service (AWS Step Functions or Prefect) to manage the flow with more visibility. Given OSBot-Utils was inspired by Prefect, we could integrate Prefect Cloud for a visual flow orchestration if desired, but it might not be necessary if our own system is robust.
- **Provenance Logging and Search:** Implement a way to easily query the provenance data. This could be as simple as a structured log or a small index that maps each article to the list of persona interests it matched. For instance, after each run, we could update a DynamoDB table or an ElasticSearch index with entries like: *(Article ID, Persona ID, MatchedEntities)*. This would allow quick look-ups to answer client queries or to debug ("why didn't this article get sent to client Y?" can be answered by checking if any overlap was found). It also allows generating analytics: e.g. how many articles were flagged for each interest area over time.
- **User Interface for Analysts:** Develop a minimal interface or utilize existing internal tools for analysts to review the outputs. This might be a simple web dashboard listing the new summaries for each persona each day, with the option to click and view details (the original article text, the extracted graph, the persona graph, and the LLM rationale). If Company X prefers, this can be done through existing content management systems or even as a CSV/email report to analysts. The goal is to make it easy for a human to spot-check the AI's work. We anticipate high accuracy in topic matching (thanks to the structured approach), but the tone of summaries or subtle prioritization might need human judgment initially.
- **Quality Assurance:** As we scale to more content and personas, we will conduct parallel runs of the old manual process and the new automated process for a period, to compare outputs. We'll gather feedback from the analysts: do the AI-tagged articles align with what they would choose? Are the summaries correct and useful? Any instances of errors (e.g. an article incorrectly matched to a persona) will be investigated. We can refine prompts or add rules to handle those. For example, if we find the LLM sometimes confuses a negative relationship ("CEO *not* involved in scandal" might erroneously tag CEO and scandal as related), we can adjust the instructions to be more precise in relationship extraction.
- **Provenance & Compliance:** Given the domains (financial, regulatory), it's crucial that our system's recommendations are explainable for compliance reasons. We will document how the provenance trail is maintained. If needed, we can integrate with any internal compliance audit systems – for instance, log every LLM call and its result to an audit log. The structured outputs make it straightforward to store these logs (each JSON is effectively a record of the AI's decision at that step).

Phase 3: Integration and Rollout:

- **Full Integration with Newsletter Workflow:** Connect the pipeline to the final delivery mechanism. If Company X delivers newsletters via email, we will generate the email content (possibly using templates where the personalized summaries are slotted in). This might involve an SMTP integration or tying into a marketing email platform's API. If the content is delivered via a portal, we will write the results into that system's database or CMS. Essentially, by this phase, no manual copy-paste should be needed; the system should feed the end-product channel directly (with an option for human approval in between, if desired).
- **Training & Handover:** Conduct training sessions for the internal team – both the technical team that will maintain the system and the analysts/editors who will work with it. The technical training will cover how the OSBot/MGraph pipeline is structured, how to adjust it (e.g. updating a prompt, adding a new data source, deploying a new version). Because the solution is built on open-source components with well-documented usage in Dinis Cruz's work, the learning curve is manageable. We will provide annotated examples and ensure the GitHub repositories (possibly Company X's fork of the OSBot and MGraph implementations) are well organized. For the analysts, training will focus on using the new tools: how to interpret the knowledge graph outputs, how to provide feedback (perhaps via a simple form or by editing a config file for taxonomy). We will emphasize that **the system is extensible** – if a new type of content or a new client need comes up, the internal team can adapt the prompts or add a new flow without needing to rebuild from scratch.
- **Human-in-the-Loop Feedback Mechanisms:** Implement any additional feedback capture needed. For instance, we could add a feature in the analyst dashboard for them to mark an entity link as "incorrect" or to merge nodes. This could simply write to a "feedback.json" that developers use to improve the ontology for the next iteration. We might also allow end-clients to rate the articles they got (thumbs up/down). Such feedback can be looped back into relevance scoring adjustments. While these are enhancements, building them now closes the loop on continuous improvement.
- **Performance and Cost Optimization:** Before full rollout, evaluate the performance of the pipeline. Thanks to serverless design, we expect it to scale on demand; however, LLM calls are the main cost driver. We will analyze how many LLM calls are being made per day (one per article for extraction, one per persona for matching, one per summary, etc.) and consider optimizations: e.g. if there are commonalities that allow batching some requests or using lower-cost models for certain steps (perhaps using GPT-4 for summary but GPT-3.5 for extraction if that proves sufficient). We will also ensure caching is effectively eliminating redundant calls.
- **Rollout and Monitoring:** Gradually roll out the automated generation for live client feeds. We might start with a subset of low-stakes newsletters to monitor performance. Implement monitoring dashboards to track the pipeline's health – e.g., number of articles processed, any errors (like LLM failing to parse something), turnaround time for each newsletter cycle. This helps demonstrate reliability and catch any issues early. Over a few

weeks, as confidence grows, we transition all personalized feed production to the new system, with analysts overseeing rather than hand-crafting each newsletter.

- **Ongoing Maintenance:** Establish a process for maintaining the knowledge graphs and prompts. As new jargon or topics emerge in the financial/regulatory world, the team will update the ontology (or the system will learn it via new examples). The modular nature of the system makes maintenance easier: for instance, if a new regulation type comes out, we might just add it to a list of entity types the extraction LLM should recognize. Or if the company enters a new vertical, we add a new persona profile and perhaps a new content source for that domain.

Leveraging Open-Source Tools and Existing Work: Throughout implementation, we will heavily reference Dinis Cruz's published work and repositories. The MyFeeds.ai project code (available on GitHub at `the-cyber-boardroom` and `owasp-sbot` repos) provides a baseline for many components. We can reuse and adapt code for the LLM prompts, the JSON schemas for entities, and the integration with MGraph. This jump-starts development and grounds our solution in a proven architecture rather than starting from scratch. The OSBot framework itself is open-source and well-suited to our needs, reducing custom development of deployment and API scaffolding. In essence, Company X's internal team will be standing on the shoulders of these open tools – gaining a solution that is *cutting-edge yet already tested in analogous scenarios*. There is no proprietary lock-in; the stack runs in Company X's cloud and the knowledge graphs and data remain fully under Company X's control.

By following this implementation plan, within months Company X can have an **automated, scalable personalized news feed system** in production. This system will dramatically reduce the manual effort required to serve each client, allow the company to scale up the number of sources and clients it can handle, and **enhance the value delivered to clients through deeper personalization and clear explanations** for each included item. Crucially, the solution is built with transparency and human governance in mind – aligning with the trust and reliability expectations in financial and regulatory domains. Together, GenAI and semantic graphs will enable Company X to maintain its competitive edge in delivering timely, relevant information to clients, while controlling the process and intellectual property through open-source innovation.