Artificial Intelligence Systems Engineering

Final Assignment

October 27, 2025

1 Introduction

OFDb is an online platform that provides film enthusiasts with a wide range of film-related content, including cast information, reviews and industry news. To achieve its organisational objectives, OFDb plans to build a new platform. A key innovation will be RecommendR, a feature for premium account holders designed to enhance the user experience by providing personalised film recommendations. Inspired by IMDb's Top Picks feature (IMDb Help Center, 2025), RecommendR will present users with their top five recommendations on the OFDb homepage, and it will be powered by a proprietary content-based recommendation system (Pazzani and Billsus, 2007). In alignment with OFDb's organisational objectives, this article defines and evaluates the system's design, deployment strategies and data considerations, and assesses risks across these domains.

2 System Goals

The goals underpinning this article are defined in full in Appendix A. In summary, OFDb seeks to achieve two primary organisational objectives: increasing advertising and premium account subscription revenue. An additional implicit objective is to minimise operating costs. Leading indicators and user outcomes serve as measurable proxies for organisational and system success, while model details provide concrete optimisation targets for the recommendation system.

For the non-AI-enabled aspects of the platform, the organisational objectives and leading indicators remain applicable. However, user outcomes require substantial adjustment, and model details are not relevant. Consequently, this article focuses on the AI-enabled components of OFDb, for which system goals are defined comprehensively.

3 System Design

The system design for OFDb places particular emphasis on the RecommendR component and its direct dependencies. The RecommendR subsystem comprises the RecommendR, User and Content servers, the FilmContent, User and FilmUserRecommendation relational databases, a vector database, the content object-store, the system telemetry subsystem and the message queue (Figure 1).

Although this subsystem forms the core of the product, it is situated within the broader system architecture, providing essential context to understand system interactions and informing subsequent discussions on deployment and data strategies.

3.1 Requirements

A set of functional and non-functional requirements for the RecommendR subsystem forms the foundation of this design, ensuring that architectural decisions remain aligned with the system's goals and constraints.

3.1.1 RecommendR Functional Requirements

1. The system shall allow users to retrieve their top five recommended films on demand.
2. The system shall allow users to request new recommendations at any time.
3. Recommendations shall be drawn from the entire catalogue of OFDb films, subject to availability and legal constraints.
4. The system shall identify, ingest and vectorise new film data at least daily.
5. The system shall allow users to apply preference filters, for example, excluding certain genres or age ratings, to their recommendations.

3.1.2 RecommendR Non-Functional Requirements

1. Performance: Users shall receive new recommendations within 1 second at the 95th percentile, regardless of geographic location.
2. Availability: The system shall be available with 99.9% uptime.
3. Model performance: The recommendation system shall maintain precision@5 $\geq$ 20% (Equation 1), monitored over a rolling one week window, and shall trigger remediation if degradation is detected.
4. Security: The system shall be protected against unauthorised access or malicious activity, both internal and external.
5. Usability: The user interface shall allow users to view and refresh recommendations with no full-page reload on desktop and mobile clients.
6. Maintainability and cost: The system shall be modular and cost-effective to operate and maintain, enabling parallel development, incremental rollout, targeted scaling, and rapid debugging.

$$\text{precision@}k = \frac{\text{True Positives among top } k}{k} \tag{1}$$

With the key requirements established, it is necessary to define the broader context within which RecommendR operates. The system context diagram illustrates the OFDb ecosystem ecosystem at a high level; its interactions with stakeholders and external data providers (Figure 2).

3.2 PACELC

The non-functional requirements prioritise availability and low latency. According to the PACELC theorem (Abadi, 2012), distributed systems must balance availability and consistency during network partitions. OFDb prioritises availability, ensuring users can always access some content, even if it is not perfectly up to date. In the absence of partitions, the system balances latency against consistency, and OFDb will favour latency to maintain a responsive user experience.

3.3 Architecture

OFDb will be hosted on Amazon Web Services (AWS), leveraging its infrastructure-as-a-service model to avoid the capital and maintenance expenditure required to directly manage hardware, networks or operating systems. AWS is a leading cloud provider, offering competitive pricing, configurability and scalability, making it an appropriate platform for both immediate deployment and future growth.

OFDb adopts a microservices architecture, decomposing system functionality into distinct, single responsibility services. Each service communicates via a remote API, typically over HTTP, and is therefore loosely coupled from other components. This separation enables horizontal scalability. For example, if the RecommendR service requires additional compute capacity, multiple instances can be deployed and load balanced to distribute requests evenly.

This architectural style also improves fault tolerance. A failure in one service, such as RecommendR, does not directly impact others. For example, the User service can continue operating independently if RecommendR fails. Additionally, microservices facilitate parallel development, allowing teams to build and deploy features independently, provided shared API contracts are in place.

However, microservices introduce challenges. Maintaining backward compatibility between evolving services can be complex, and the overhead of remote communication will introduce additional latency. By contrast, a monolithic architecture avoids these issues, but is more difficult to scale, less resilient to failures and harder for multiple teams to develop concurrently.

Certain design choices, such as decomposing functionality into multiple microservices, are more expensive than the simplest possible alternative, but they provide long-term benefits in scalability, maintainability and developer velocity. These decisions should be evaluated holistically; if a design is financially feasible, it is often better to invest in a robust, low-debt architecture early, rather than accumulate technical debt that will be more expensive to resolve later.

Given OFDb's ambitious latency and availability targets, and the need for scalability, maintainability and fault tolerance, the microservices model, when paired with an appropriate deployment strategy, represents a balanced and industry standard choice. An enumeration of the key principles for software architecture design and how this system embodies them can be found in Table 1.

Each service will be deployed as a Docker container within a Kubernetes network. Docker provides lightweight containerisation, isolating applications within virtualised Linux environments, while Kubernetes orchestrates and manages these containers across the system. This approach simplifies environment setup, accelerates development and deployment, and enables on-demand scalability. OFDb will use AWS's Elastic Kubernetes Service (EKS), a managed container-as-a-service platform, to streamline deployment and operations.

OFDb users will interact with the platform through a single-page application built with Angular, a web application framework developed and maintained by Google. Angular's component based architecture and mature tooling make it an industry standard choice for developing rich, interactive web applications and provide a robust foundation for OFDb's frontend.

To minimise latency for OFDb's global user base, content delivery networks (CDNs) will serve static assets, such as images, film metadata, and web resources, from servers geographically close to users. This ensures responsive performance and supports the system's latency requirements, and AWS's CloudFront is a natural initial choice.

The Gateway server acts as the backend entry point, providing both authentication and load balancing. Authentication will be managed by Auth0, a specialist provider enabling secure and seamless login via social accounts. Outsourcing this critical function ensures industry standard security while eliminating the complexity of maintaining a self-managed authentication system. Nginx will serve as the load balancer, using the least-connected configuration to route incoming requests to the server with the fewest active connections, optimising throughput and reliability (NGINX, Inc., 2025). The Gateway server will also enable traceability throughout the whole system by generating a unique transaction id for each request, with each system component including it in every application log. The id will propagate across services, enabling engineers to retrospectively trace a single request through the system, which is vital for debugging issues.

A backend-for-frontend (BFF) server will mediate all communication between the UI and backend microservices. This layer exposes a unified API, simplifying frontend integration, accelerating development and improving long-term maintainability.

The User server manages operations related to user data, such as retrieving and updating personal details and film preferences, by interfacing with the UserDb relational database. AWS RDS for PostgreSQL is the leading candidate for a cloud relational database solution.

The Content server performs equivalent functions for application content, including film metadata, reviews, and images. It employs a hybrid storage solution combining the ContentDb relational database table with object-based file storage, such as AWS's Simple Storage Service. In this model, table rows reference corresponding file locations, maintaining good query performance while significantly reducing storage costs for large media assets.

This hybrid approach is particularly important given OFDb's commitment to maintaining a rich, enduring catalogue of film content spanning multiple decades; the architecture ensures scalability and cost efficiency.

Together, the User and Content services provide the foundational data required by RecommendR, which uses this information to generate feature vectors for user and film entities. Recommendations are drawn from the full OFDb catalogue, with optional user-defined filters, such as genre or age rating, to refine results.

Each vector combines learned embeddings, calculated using a library like SentenceTransformers (Reimers, Gurevych, and contributors, 2025), to capture semantic relationships, with hand-crafted features that represent explainable, domain specific attributes. These vectors are stored in a vector database such as AWS Pinecone, and recommendations are generated through an approximate nearest neighbour search (Pinecone Systems, Inc., 2025). This transforms the search into a high performance similarity query, reducing latency. Filters are then applied to exclude dismissed or already seen recommendations and to respect user preferences, and the UserFilmRecommendation table is updated with the latest recommendations. Using a managed vector database simplifies the architecture, improves reliability and lowers maintenance costs.

A Kafka-based message queue, such as AWS's Managed Streaming for Apache Kafka, enables event-driven communication between the RecommendR, User, Content and Scheduler servers. Kafka's publish–subscribe model supports asynchronous workflows and robust data flow, while its commit log and replication features ensure reliability and fault tolerance.

Feature vectors are generated on an event-driven basis for efficiency and scalability. A Scheduler service emits a Kafka event periodically to load new film content, and the Content server subsequently emits an event to RecommendR when new vectors are ready to be calculated. User triggered events, such as a new user joining the platform, emit Kafka events in real time to generate the user's feature vector immediately.

Telemetry data is ingested into Snowflake, a cloud based data warehousing platform well suited to OFDb's analytical and scalability needs. System components send telemetry directly to Snowflake, where it becomes queryable within minutes (Snowflake Inc., 2025). Its compute layer enables large scale aggregation and analysis, providing behavioural and performance insights, such as user engagement with the RecommendR recommendations.

System health monitoring is achieved through Grafana, backed by a Prometheus time-series database. Application servers emit timer and counter metrics to the Telemetry server, which in turn populates Prometheus, allowing engineers to assess system performance in near real time. Automated alerts can be configured for key thresholds, for example, sending a notification if BFF server latency exceeds 1000ms, ensuring rapid response and sustained reliability across the platform.

Application logs will be published from each container to a centralised log aggregation platform, such as Splunk. In Splunk, logs can be queried using the Splunk Search Processing Language, enabling engineers to investigate issues across multiple services and correlate them with relevant telemetry signals. For example, if Grafana alerts on elevated request latency, engineers can query Splunk to retrieve the corresponding application logs for the affected service and timeframe, quickly identifying the root cause.

3.4 Data Models

Adhering to normal form design principles is best practice because it minimises data redundancy, enhances data integrity and reduces the likelihood of anomalies. This structure ensures that relationships between users, films and recommendations remain consistent and efficiently queryable. Figure 3 shows the normal form relational database tables of User, FilmContent, UserFilmRecommendation and Content, as well as how they interact with the vector database. Figure 4 shows the interaction between components of the OFDb system and the system telemetry subsystem. Figure 5 shows the interaction between the UI and the user telemetry subsystem.

4 Risks

OFDb's system is exposed to a range of risks arising from incorrect requirements, assumptions and specifications, as well as changes in its operating environment (Jackson, 1995). These risks threaten the organisational objectives and must therefore be anticipated and mitigated through system design.

The system goals are interrelated, though the strength of these relationships vary. They can be viewed as an n-ary tree, beginning with high level organisational objectives and narrowing through increasingly concrete and measurable goals, down to model details. Figure 6 presents a diagram inspired by fault tree analysis (Lee et al., 2009), in which each system state block and primary failure event is the inverse of a proposed goal. The diagram also highlights anticipated risks at multiple levels, reflecting the reality that objectives may be undermined by a range of internal and external factors.

Following will be an in-depth look into each of the primary failure events, both directly and indirectly, assessing risks both caused and implied by incorrect predictions.

4.1 Poor Prediction Performance

RecommendR can be considered underperforming if its continuously monitored precision@5 $< 20\%$. Low model performance implies a poor user experience. If users are shown films they do not want to watch, false positives, or are not shown films they would watch, false negatives, they are less likely to use the recommendation feature.

Poor model performance propagates up the goal hierarchy described in the modified fault tree analysis diagram. If users perceive the recommendations as unhelpful, overall sentiment toward the platform will decline, increasing the likelihood that premium users cancel their subscriptions. This directly endangers the organisational objective of increasing premium account revenue. Poor predictive performance also threatens advertising revenue. If OFDb cannot demonstrate that its recommendation system is effective, advertisers will be less likely to pay to promote their content to targeted users.

Poor prediction performance represents a major risk because it directly endangers the two primary organisational objectives, but it can be mitigated in several ways. For example, the model should be optimised to meet the defined model detail goals (Appendix A), in particular ensuring that precision@5 $\geq 20\%$ at release on a representative synthetic test set. Second, model performance should be continuously monitored via telemetry, with remedial action taken if the performance drops (see MLOps subsection).

False positives can be reduced by incorporating informative hand-crafted features and by correctly enforcing user preference filters, for example, by not recommending excluded genres. False negatives can be addressed with the same feature engineering approach, alongside ensuring that the model has access to a sufficiently broad catalogue of films. Given the international reach of the platform, the most direct way to support this issue is to maintain as complete a catalogue of films as possible.

4.2 Trust, Explainability and Regulatory

A consistently underperforming model erodes trust and suggests weak explainability; if its decisions were better understood, performance could likely be improved. Yet trust extends far beyond model accuracy alone. Users entrust OFDb with their personal data, time and money. If that trust is compromised, all organisational goals are at risk.

Trust is shaped by how safely users can participate in the platform. OFDb must act to prevent behaviours that undermine users' sense of safety and dignity, including harassment, cyberbullying and the uploading of inappropriate or harmful media. Tolerating such behaviour would, in extreme cases, expose the organisation to legal risk. Moderation, escalation and enforcement processes therefore play as much a role in maintaining trust as technical performance does.

OFDb's primary markets are North America and Western Europe, regions with strong expectations around human rights and ethical business practices. It is therefore essential that OFDb, along with all downstream suppliers and partners, upholds rigorous ethical and compliance standards. Any breach of these laws and principles, or even the perception of a breach, would not only undermine user confidence, but could also expose OFDb to financial, legal and reputational consequences.

RecommendR, and OFDb's broader ecosystem, must work proactively to build and sustain user trust by maintaining a high standard of quality, transparency and responsibility, and by continually striving to exceed it. Various strategies for how the system can support this objective are outlined in the Deployment Strategies and Data sections.

4.3 Sustainable Data Source

Data procurement introduces an additional technical risk vector. OFDb must secure a reliable, sustainable supply of the data required to power RecommendR, and must maintain the infrastructure and expertise needed to manage that data over time. If high-quality film content data is not consistently available, recommendation quality will degrade. This risk can be mitigated by integrating with at least two independent data providers for each category of system-critical data, ensuring redundancy and reducing reliance on a single source.

4.4 Algorithmic Bias, Privacy and People

Processing user information in RecommendR introduces privacy risk because the content-based model may implicitly learn sensitive attributes such as protected characteristics or personal preferences. Any leakage or misuse of this data, for example, unauthorised disclosure or sale to third parties, would create significant legal, ethical and reputational risk.

Collecting user telemetry introduces privacy concerns. Telemetry such as click-through rate, refresh behaviour, and satisfaction surveys can reveal viewing habits, taste profiles, inferred demographics and, potentially, sensitive preferences. Even if this data is not explicitly labelled as personal, it can become indirectly identifying in aggregate over time. Under the General Data Protection Regulation, OFDb bears a legal and ethical responsibility to protect users' personal data, process it transparently and ensure it is used only for clearly defined purposes (GDPR, 2016). Therefore, telemetry used for any purpose should be aggregated and anonymised, and retention should be limited to what is strictly necessary.

There is also legal risk arising from bias in recommendations. For example, RecommendR may disproportionately surface films with predominantly ethnic minority casts only to users with non-Western names. This behaviour could unfairly disadvantage certain films or audiences along protected characteristics, creating regulatory and reputational risk. In addition, film studios may attempt to game the system by manipulating metadata so that their titles are recommended more often, which would erode recommendation integrity and undermine commercial trust.

A lack of in-house machine learning (ML) expertise can lead to the introduction of unintended algorithmic bias. Misaligned incentives may also encourage behaviours that harm OFDb's long-term objectives. For example, ML engineers may be incentivised to optimise model details system goals at the expense of user privacy.

These risks can be mitigated by dedicating a team to establish and govern OFDb's data and privacy policies. These policies can form the basis of robust training, oversight and accountability practices, and may be directly realised by employing an independent team to monitor system behaviour, auditing for bias and the problematic use of data, enforcing controls when issues are detected.

5 Deployment Strategies

The system consists of containerised microservices running on dedicated compute, as opposed to running on a function-as-a-service platform, for example, AWS Lambda. This supports cloud-provider agnosticism: OFDb can migrate to another provider or to on-premises infrastructure with less friction, reducing vendor lock-in. This is a deliberate trade-off. While tighter integration with AWS services could be cheaper initially, greater independence reduces strategic risk over time.

A key benefit of using a cloud provider is the flexibility afforded. For example, RecommendR does not require specialised hardware. Although GPUs are commonly used for large-scale model training due to their parallelism, the content-based approach used here is not computationally intensive, so standard CPU instances are sufficient. If this assumption changes, OFDb can still leverage AWS offerings such as SageMaker for demanding model training.

5.1 Geographic Considerations

OFDb is a globally available platform with an international user base, and its services must therefore remain accessible regardless of geographic location. Network latency varies significantly by region, for example, AWS reports an average latency of $230ms$ between the us-west-1 and ap-south-1 regions, a proxy for a user in India connecting to U.S. servers, compared with $75ms$ between us-west-1 and us-east-1 (CloudPing, 2025). As outlined in the System Design section, CDNs play a key role in ensuring low-latency delivery of static assets such as images and web resources. However, other parts of the system, particularly those involving real time computation, must also account for geography.

To minimise latency and distribute load, OFDb will deploy AWS EKS clusters in North America, Western Europe, and Central Asia, regions corresponding to its largest user populations. This ensures user requests are routed to a nearby compute cluster rather than across continents, improving response times and providing a natural form of load balancing. A drawback of this geographic segregation is increased data fragmentation: cross-region queries become more complex, and users travelling internationally may experience inconsistent performance. Furthermore, maintaining an internationally distributed system will be more costly, both in terms of capital expenditure and ongoing maintenance. A global footprint also suggests a follow-the-sun support model, ensuring critical issues receive prompt attention around the clock. However, this approach typically requires engineering teams in multiple time zones, which may not align with OFDb's current operational strategy or resource constraints; geographically distributed clusters enhance scalability but increase financial and operational complexity. For example, if OFDb is used within the European Union (EU), it would be subject to the EU AI Act, which imposes strict regulatory requirements and significant penalties for non-compliance (EU AI Act, 2024). All the aforementioned factors illustrate the trade-off between reducing latency and increasing system and operational complexity.

Finally, OFDb must recognise that market differences may affect model performance. For example, preferences within Bollywood cinema may differ markedly from those in Hollywood, leading to differing recommendation dynamics. The extent and significance of these differences should be validated empirically using model performance metrics and cash-flow analysis, to ensure that technical decisions align with the organisational objectives. If significant, OFDb should pursue one of two strategies:

1. Generalised model: a single, globally deployed model optimised for broad applicability, trading precision for maintainability.
2. Market-specific models: tailored models for major regions, improving recommendation quality at the expense of higher development and maintenance costs.

5.2 Release Strategy

When releasing new features, risks must be minimised, whether from software defects or poor user reception. Reliability, stability, and performance are first verified through canary testing, in which a small, periodically increasing user subset receives the new feature to confirm its operational safety and stability.

Once validated, A/B testing is used to assess feature performance and user sentiment by comparing a test group with a control group. For example, when deploying an updated recommendation algorithm, OFDb may expose 10% of randomly assigned users, sampled by region, to a new vector database instance, while the remaining users continue using the existing one. Feature flags in the code ensure that only the sampled users see the new behaviour, and the assignment strategy ensures minimal bias while encoding proportional representation from each operating region.

Effectiveness is evaluated using telemetry and user surveys, for example, click-through rate and satisfaction scores (see Data section). Statistical significance is determined using Welch's two-sample t-test (Welch, 1947), and practical impact is measured via effect size, calculated as the difference in the groups' mean values. However, A/B tests have known pitfalls related to the human psychology of being part of an experiment, and these should be mitigated by statistics specialists in experiment design and execution (Kästner, 2021).

While essential, A/B testing introduces versioning and deployment complexity; multiple live variants of models, databases and services must interoperate seamlessly. Managing this complexity relies on configuration management and infrastructure as code, which makes infrastructure versioned, transparent and easily reversible. Services such as AWS EKS further abstract deployment complexity, though effective version management, DevOps/MLOps discipline and backwards-compatible updates remain crucial to ensuring reliability as components evolve independently.

5.3 DevOps

DevOps considers the development and operation of a software system to be tightly coupled (Ebert et al., 2016). Implementing robust DevOps practices and tooling ensures that OFDb delivers high-quality, maintainable software efficiently and consistently. The DevOps lifecycle for OFDb encompasses planning, creation, verification, packaging, release, configuration and monitoring.

All development begins with structured planning and issue tracking. Source code will be version controlled using Git, enabling engineers to document every change as a commit with descriptive metadata. Branching strategies will isolate development, testing and production environments, supporting parallel workstreams and safe integration. Using Git integrated with a project management platform such as Microsoft's Azure DevOps, OFDb can link work items directly to code commits, ensuring transparency and traceability between business goals, tasks and technical implementation. OFDb will follow an agile working philosophy, so milestones can be defined to align sprint objectives with organisational priorities, allowing data driven progress tracking across the system goals hierarchy.

Automated testing is the foundation of continuous integration (CI). Unit, integration and system tests will be executed automatically within the CI pipeline, ensuring new commits do not break existing functionality. Test coverage should exceed 85%, and static code analysis tools like SonarQube will further verify code quality, compliance with security and performance standards before deployment.

Once verified, each service will be packaged into a versioned Docker image. These artifacts will be stored in Artifactory or an equivalent repository to preserve all current and historical versions, allowing full traceability and rollback to any prior release. This guarantees reproducibility and controlled deployment across environments.

Continuous deployment (CD) pipelines will conditionally promote validated artifacts through staging and into production. Each release will be auditable, ensuring accountability and reproducibility. Rollbacks can be executed efficiently by redeploying a previous Docker image from the repository.

Infrastructure-as-code will be used to define and version control all infrastructure specifications, enabling transparent, repeatable environment provisioning. This config-driven approach enables consistent configuration across clusters and regions.

Comprehensive monitoring completes the lifecycle. System telemetry will be collected through Prometheus and visualised in Grafana (see Data section). Automated alerts can trigger scaling actions in EKS when thresholds are exceeded, maintaining performance while minimising cost through elastic scaling. Application logs are collected and made accessible in Splunk (see Data section).

5.4 MLOps

MLOps introduces additional concerns beyond DevOps because changes to data, features, and models can alter system behaviour even when application code is unchanged. For RecommendR, the focus is on four areas: CI, CD, continuous monitoring, and data/feature lifecycle management.

CI for RecommendR must validate not only code but also data, schemas and feature generation. A dedicated pre-production environment will be used to train and evaluate updated versions of the recommendation logic. This environment will operate on a controlled synthetic dataset representing typical user–film pairs. The dataset is used to generate a benchmark precision@5 score, enabling comparison between feature vector revisions.

As part of CI, new feature extraction code must run successfully on historical data samples. Schema validation must ensure backward compatibility, so downstream services are unaffected. Test coverage for feature generation code should meet or exceed 85%, consistent with the broader DevOps standard, ensuring that changes to data processing are treated as rigorously as changes to application logic.

CD for RecommendR is lightweight compared to some ML systems because the model is largely deterministic and explainable. There is no bespoke trainable model being continually re-deployed, and therefore there is no need for automated continuous training in production. The deployment of changes primarily affects the feature vector construction pipeline and the vectors stored in the vector database.

Rolling out an update, for example, a change in feature composition, can follow the same release path as standard application code: build, package into a versioned container, deploy to the pre-production environment, then promote to production. Where user facing ranking behaviour is expected to change, A/B testing is used (see Release Strategy subsection).

Continuous monitoring is critical, and two classes of risk must be monitored over time: outliers and model drift (Klaise et al., 2020). Models can fail on inputs outside the training data distribution; outlier detection helps flag inputs or recommendations that fall outside expected behaviour, and libraries such as Alibi Detect can be integrated to detect anomalous requests (Seldon Technologies LTD, 2024). Drift occurs when the relationship between users and films changes over time. OFDb will track drift by:

1. Re-scoring the synthetic test set and comparing the current precision@5 against the historical benchmark.
2. Comparing distributions of user telemetry such as click-through rate and thumbs-down rate, over time using Welch's two-sample t-test (see Release Strategy subsection).

A significant drop in precision@5, or a sustained decline in user sentiment toward recommendations, indicates that the current feature representation and filtering assumptions are no longer valid and require remediation, and may be an indication of concept drift (Pham et al., 2025). Telemetry and survey data used for this monitoring are described in the Data section.

A key operational challenge is managing the evolution of feature vectors over time. Two strategies can be employed:

1. Incremental enrichment: new users and films receive vectors generated with the latest feature logic, while existing vectors are left unchanged. This is simple and avoids reprocessing the full catalogue, but can create inconsistency if old and new vectors are not directly comparable.
2. Full regeneration: all user and film vectors are rebuilt using the latest embeddings and feature construction logic, then bulk reloaded into the vector database. This yields global consistency and cleaner similarity search, but requires a coordinated migration (and may require temporarily serving two indexes during A/B testing).

OFDb can choose per-change which strategy to apply based on expected impact. Minor feature tweaks may justify append-only updates; major representational changes should trigger a full regeneration.

To support reliable evaluation across these updates, the synthetic dataset must itself be maintained. This dataset should be curated, versioned and periodically reviewed to ensure it still reflects real usage patterns. K-fold cross validation can be used during testing to maximise the utility of a relatively small labelled dataset (Kohavi, 1995). The precision@5 measured on this dataset should then be compared with observed user engagement, for example, click-through rates from live A/B tests, to confirm that offline metrics remain predictive of real outcomes.

5.5 Human Ownership and AutoML

AutoML platforms such as AWS SageMaker Autopilot can automate parts of the ML workflow, including data preparation, model selection and hyperparameter tuning. OFDb is not relying on AutoML in the initial system for two reasons. First, RecommendR is deliberately simple, explainable and largely hand engineered to support transparency and trust. Second, OFDb must retain in-house expertise; overdependence on automated tooling could create vendor lock-in and weaken organisational understanding of model behaviour, bias and legal exposure. AutoML may be used as an accelerator or exploratory aid, but it does not replace OFDb's responsibility to understand, justify and govern the recommendation logic internally.

6 Data

The data handled by the OFDb system can be divided into three primary categories: user data, content data, and telemetry data. User data encompasses information specific to individual users, including personal details, account configurations and film preferences. Content data refers to material used by the system to inform the RecommendR model or enhance the user experience, such as film clips, images, news articles, reviews and metadata. For the purposes of this section, the primary focus is on telemetry data, which captures information about the behaviour and performance of users and of the system itself. Telemetry can be further subdivided into two categories: system telemetry and user telemetry. All user telemetry is recorded in the context of its operating conditions. For example, if a user is part of an A/B testing, the feature variant their telemetry corresponds to will be recorded as part of the telemetry. This allows accurate retrospective analysis of telemetry for different aspects and versions of the system. Table 2 defines all telemetry expected to be captured across the system.

6.1 System Telemetry

System telemetry refers to metrics collected from the various components of the OFDb system, providing continuous insight into their health and performance. It is essential for verifying that non-functional requirements are met over time. For example, request latency will be monitored to ensure that the 95th percentile latency remain within expectations.

Telemetry data will be stored in a time-series Prometheus database and visualised through tools such as Grafana, allowing for programmatic access when required. The selected metrics are chosen to provide deep operational visibility from a component perspective, enabling engineers to assess system health, identify anomalies and take proactive measures. For example, a Grafana alert could be configured to trigger if a container's memory usage exceeds 80%, automatically notifying the Kubernetes control plane to deploy an additional instance.

System telemetry also supports reliability analysis by tracking component level failure rates, measured as the proportion of time a service is unexpectedly unavailable. These data enable the creation of reliability block diagrams, which map interdependencies between components and deriving reliability scores.

Complementing telemetry, application logging plays a crucial role in diagnosing issues and understanding system behaviour in greater detail. While telemetry provides aggregated, quantitative insight, logs capture the discrete events, stack traces and contextual information essential for root-cause analysis. All application and infrastructure logs will be persisted to ensure that transient or container-scoped logs are not lost during redeployments or failures.

6.2 User Telemetry

User telemetry captures behavioural and interaction data from users engaging with the OFDb platform. While system telemetry focuses on the status of system components, user telemetry provides insight into how effectively the system, and particularly the RecommendR subsystem, serves its users. Metrics collected through user telemetry can be divided into two categories: direct metrics and proxy metrics.

Direct metrics represent user actions that explicitly convey satisfaction or dissatisfaction with recommendations. For example, the click-through rate (CTR) on recommendations serves as a direct indicator of engagement and, in combination with recommendation refresh telemetry, can be used as a proxy for precision@5. A consistently low CTR may suggest an overabundance of false positives or missed relevant items, false negatives. Similarly, the "thumbs down" feature enables users to explicitly reject unsuitable recommendations, producing structured negative feedback that can be aggregated to assess model performance, in particular implying a false positive recommendation.

Proxy metrics provide indirect evidence of user experience and system quality. For example, frequent recommendation refresh requests may imply user dissatisfaction, although such actions must be interpreted with caution as they may reflect curiosity or exploratory behaviour. Recommendation focus, the amount of time a user spends looking at a currently recommended film, is also captured as a proxy for recommendation relevance. Prolonged inactivity or an absence of engagement with RecommendR could suggest a lack of relevant recommendations, but these inferences remain probabilistic and must be treated as secondary indicators.

In addition to telemetry-based metrics, periodic user surveys will be integrated into the platform to capture subjective sentiment and qualitative insights that cannot be inferred from behavioural data alone. Short, optional surveys, such as rating satisfaction with recent recommendations or perceived relevance of content, provide a valuable addition to quantitative telemetry. Survey responses will be anonymised, stored alongside user telemetry in Snowflake and analysed in aggregate to identify trends in perceived recommendation quality and overall platform sentiment. Over time, survey derived sentiment scores can be correlated with objective engagement metrics, offering a view on how well the system is achieving OFDb's user outcomes and leading indicators.

At scale, these telemetry events enable statistically significant trend analysis. Given OFDb's large and diverse global user base, even small behavioural variations can yield meaningful insights within short timeframes. For example, with approximately $1,000,000$ active weekly users, user telemetry may generate hundreds of thousands of events per week. Telemetry events will generally remain under $1\,KB$ in size and users are expected to generate an average of 20 telemetry events across all categories in a week. Therefore, a crude estimate for the weekly data size of user telemetry, not including application logs, is:

$$\frac{1{,}000{,}000 \times 20 \times 1\,\mathrm{KB}}{1{,}048{,}576\,\mathrm{KB/GB}} \approx 19\,\mathrm{GB} \tag{2}$$

Even if OFDb decides to process an entire year's worth of telemetry in batch, the total data size will be of the order of 1 TB, a comfortably manageable quantity given the scalability of Snowflake's elastic compute. System telemetry is recorded in significantly higher volumes. In terms of data size, timers, counters and gauges are very small, but application logs can vary significantly. Given the significant variation in volume, size is not estimated here. All telemetry data will be stored for a minimum of 12 months to build a rich dataset for evaluating model performance, conducting A/B testing, validating feature changes and enabling historical debugging. The latter can be especially important if, for example, a bug is released into production but only recognised months later.

Finally, user telemetry also supports operational decision making. The number of active users and concurrent active sessions per region can be monitored to inform horizontal scaling strategies, ensuring low latency and consistent user experience across global deployments.

7 Conclusion

Film news and recommendation platforms are a mature, well understood class of system, and OFDb is already established in the market. In that sense, introducing a new platform centred around RecommendR is not without precedent. However, when designing a new product, nothing should be taken for granted. Assumptions about performance, compliance, user experience and cost must be made explicit and designed for.

This article has proposed a system architecture intended to meet OFDb's organisational goals: increasing premium revenue, increasing advertising revenue and minimising operating cost. The recommended design adopts a microservices architecture with clear separation of concerns, versioned deployment and infrastructure-agnostic deployment on containerised compute. This approach, together with adherence to core software architecture principles, such as modularity, abstraction and loose coupling, provides a foundation for a system that is scalable, extensible, maintainable and performant.

The article has also identified and analysed a wide range of risks. These include model underperformance, bias, data availability, user safety, privacy and compliance obligations, operational failure, geographic scaling challenges and misaligned incentives within the organisation. For each, mitigating strategies were proposed: continuous monitoring of recommendation quality and user sentiment, access control and data minimisation, bias auditing, maintaining redundant data sources, content moderation and preserving inhouse expertise.

Deployment and continuous operation were treated as first class design concerns; the proposed deployment strategy explicitly considers multi-region delivery, availability and latency trade-offs and regulatory variation across jurisdictions. A comprehensive release strategy is also defined, including canary testing for reliability and stability, followed by A/B testing to measure user impact using telemetry, surveys and statistical validation. Long-term operation is supported by thorough DevOps and MLOps practices, including infrastructure as code, versioned artifacts, reproducible environments, pre-production evaluation on synthetic benchmarks, drift and outlier monitoring, centralised telemetry, and behavioural user telemetry alongside application logs in Snowflake and Splunk.

Finally, the article categorised the data handled by the system and described how telemetry can be used responsibly to improve the platform. System telemetry supports observability and resilience. User telemetry and user surveys enable continuous measurement of recommendation quality and user satisfaction, informing iterative improvement while highlighting emerging drift or sentiment risks.

In conclusion, this article serves as a high level blueprint for delivering OFDb's new platform; recognition and mitigation of all predictable risks, paired with an architecture intended to scale and an operational model intended to last.

References

Abadi, D., 2012. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *Computer*, 45(2), pp.37–42.

Artificial intelligence act (regulation (eu) 2024/1689 of the european parliament and of the council of 13 june 2024 laying down harmonised rules on artificial intelligence and amending certain union legislative acts) 2024 [Online] ((EU) 2024/1689). Accessed 27 Oct 2025. Available from: `https://eur-lex.europa.eu/eli/reg/2024/1689/oj/eng`.

CloudPing, 2025. *CloudPing* [Online]. Available from: `https://www.cloudping.co/`.

Ebert, C., Gallardo, G., Hernantes, J., and Serrano, N., 2016. DevOps. *Ieee software*, 33(3), pp.94–100.

General data protection regulation (regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data) 2016 [Online] ((EU) 2016/679). Accessed 27 Oct 2025. Available from: `https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng`.

Hulten, G., 2018. Defining the intelligent system's goals. In: *Building intelligent systems: a guide to machine learning engineering* [Online]. Berkeley, CA: Apress, pp.35–49. Available from: `https://doi.org/10.1007/978-1-4842-3432-7_4`.

IMDb Help Center, 2025. *What to watch FAQ* [Online]. Available from: `https://help.imdb.com/article/imdb/discover-watch/what-to-watch-faq/GPZ2RSPB3CPVL86Z#`.

Jackson, M., 1995. The world and the machine. *Proceedings of the 17th international conference on Software engineering*, pp.283–292.

Kästner, C., 2021. *Machine learning testing and experimenting in production* [Online]. Available from: `https://ckaestne.medium.com/quality-assurance-in-production-for-ml-enabled-systems-4d1b3442316f`.

Klaise, J., Van Looveren, A., Cox, C., Vacanti, G., and Coca, A., 2020. *Monitoring and explainability of models in production.* Unpublished. arXiv: `2007.06299`.

Kohavi, R., 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. *Proceedings of the 14th international joint conference on artificial intelligence - volume 2*, IJCAI'95. Montreal, Quebec, Canada: Morgan Kaufmann Publishers Inc., pp.1137–1143.

Lee, W.-S., Grosh, D.L., Tillman, F.A., and Lie, C.H., 2009. Fault tree analysis, methods, and applications a review. *Ieee transactions on reliability*, 34(3), pp.194–203.

NGINX, Inc., 2025. *NGINX HTTP load balancing* [Online]. Online documentation. Available from: `https://nginx.org/en/docs/http/load_balancing.html`.

Pazzani, M.J. and Billsus, D., 2007. Content-based recommendation systems. In: *The adaptive web: methods and strategies of web personalization.* Springer, pp.325–341.

Pham, T.M.T., Premkumar, K., Naili, M., and Yang, J., 2025. *Time to retrain? Detecting concept drifts in machine learning systems* [Online]. arXiv: `2410.09190 [cs.LG]`. Available from: `https://arxiv.org/abs/2410.09190`.

Pinecone Systems, Inc., 2025. *Pinecone algorithms set new records for BIGANN* [Online]. Available from: `https://www.pinecone.io/blog/pinecone-algorithms-set-new-records-for-bigann/`.

Reimers, N., Gurevych, I., and contributors, t.S.-T., 2025. *Sentence-transformers documentation* [Online]. Available from: `https://www.sbert.net/`.

Seldon Technologies LTD, S.T., 2024. *Alibi-detect: Algorithms for outlier, adversarial and drift detection* [Online]. Available from: `https://github.com/SeldonIO/alibi-detect`.

Snowflake Inc., 2025. *Introduction to snowpipe* [Online]. Available from: `https://docs.snowflake.com/en/user-guide/data-load-snowpipe-intro`.

Welch, B.L., 1947. The generalization of 'student's' problem when several different population variances are involved. *Biometrika* [Online], 34(1/2), pp.28–35. JSTOR: `2332510`. Available from: `http://www.jstor.org/stable/2332510` [Accessed October 26, 2025].

A System Goals

A.1 Organisational Objectives

The consideration to build RecommendR is principally motivated by OFDb's organisational objectives (Hulten, 2018); to increase revenue generated through advertising and premium account subscription fees by 5% and 10%, respectively. Both goals can be measured by calculating the total revenue generated per reporting period for each income stream and comparing to previous financial statements. While these are meaningful goals for OFDb as an organisation, they are not useful goals for the teams implementing RecommendR for three reasons. Firstly, advertising and premium user revenue are only loosely coupled to the performance of RecommendR, thus the system cannot be optimised based on these goals. Secondly, they are slow indicators with significant lead time between implementation and measurement. Thirdly, they are affected by many extrinsic factors, such as market forces (Hulten, 2018). Therefore, the correlation between these goals and a successful implementation of RecommendR is likely to be positive but weak.

A.2 Leading Indicators

Leading indicators are correlated with future success, and the two main indicators are user sentiment and user engagement (Hulten, 2018). OFDb plans to assess sentiment directly via a program of regular, digital surveys, and indirectly via A/B testing, designed to test new features against a baseline. On RecommendR's release, OFDb aims to achieve 90% positive sentiment across the user base for both sentiment measures. This measure can act as a proxy for how likely the users are to keep their subscription, thus relating to the broader organisational goal of increasing premium account revenue. User engagement gauges how much the customers use the product and can be measured by counting the average number of interactions with the website each user makes per day. OFDb expects a 5% increase in user engagement post RecommendR's release. Measuring user engagement can act as more than a passive indicator; it may directly contribute to the organisational objectives by being used as evidence to secure more lucrative advertising contracts, by proving that OFDb has an engaged and broad user base.

A.3 User Outcomes

User outcomes focus on setting goals pertaining to the user experience, and as such often relate strongly to model implementation decisions. Firstly, OFDb expects recommendations to be up to date, with a maximum wait of one day between breaking news and recommendation updates. Secondly, OFDb expects that, for each set of five film recommendations, at least one should be a film the user would watch next. It may be difficult to obtain high quality and consistent feedback from users on the performance of RecommendR. Therefore, investing in building a supervised test suite with examples covering a wide spectrum of user and recommendation combinations is advisable. As well as contributing to increasing premium account revenue, user sentiment and user engagement, a successful implementation of this user outcome links strongly to increasing advertising revenue. If OFDb can demonstrate accurate recommendation performance, then prospective advertisers will be more likely to trust the platform to recommend relevant advertisements to the users.

A.4 Model Details

Constructing goals around model details is valuable because they can be used to directly optimise the model. The model should be optimised to account for the different kinds of errors a user may experience: false positives and false negatives. OFDb has decided that, if necessary, it is acceptable to trade recall for precision. In the extreme case, supplying no recommendations will provide a worse user experience than providing poor recommendations; providing none may give the impression that RecommendR is broken. Therefore, the model will be optimised to have a precision@5 $\geq 20\%$, with the final metric chosen to align with the prior stated user outcome.

The RecommendR system must also be architected to be able to account for daily updates, such as new film announcements and other metadata changes, while maintaining the mandated precision@5 score. This will ensure a good user experience and further alignment with the user outcomes.

The average cost of recommendation inference per user, measured as the cost of compute and resources required to generate a recommendation averaged across users monthly, must be no greater than a fraction of the premium account subscription fee, to be determined by a future cash flow analysis. This will maximise any post-release profit margin increases, but may result in a more complex system architecture, potentially resulting in a larger up-front investment.

Model details goals do not capture the user's holistic system experience, because the goals focus on optimising aspects of low-level system architecture and performance. Therefore, considering these goals in partnership with leading indicators and user outcomes is critical to anticipating whether the system is likely to positively contribute towards the organisational objectives.

B Architecture Diagrams

B.1 Architecture Diagram

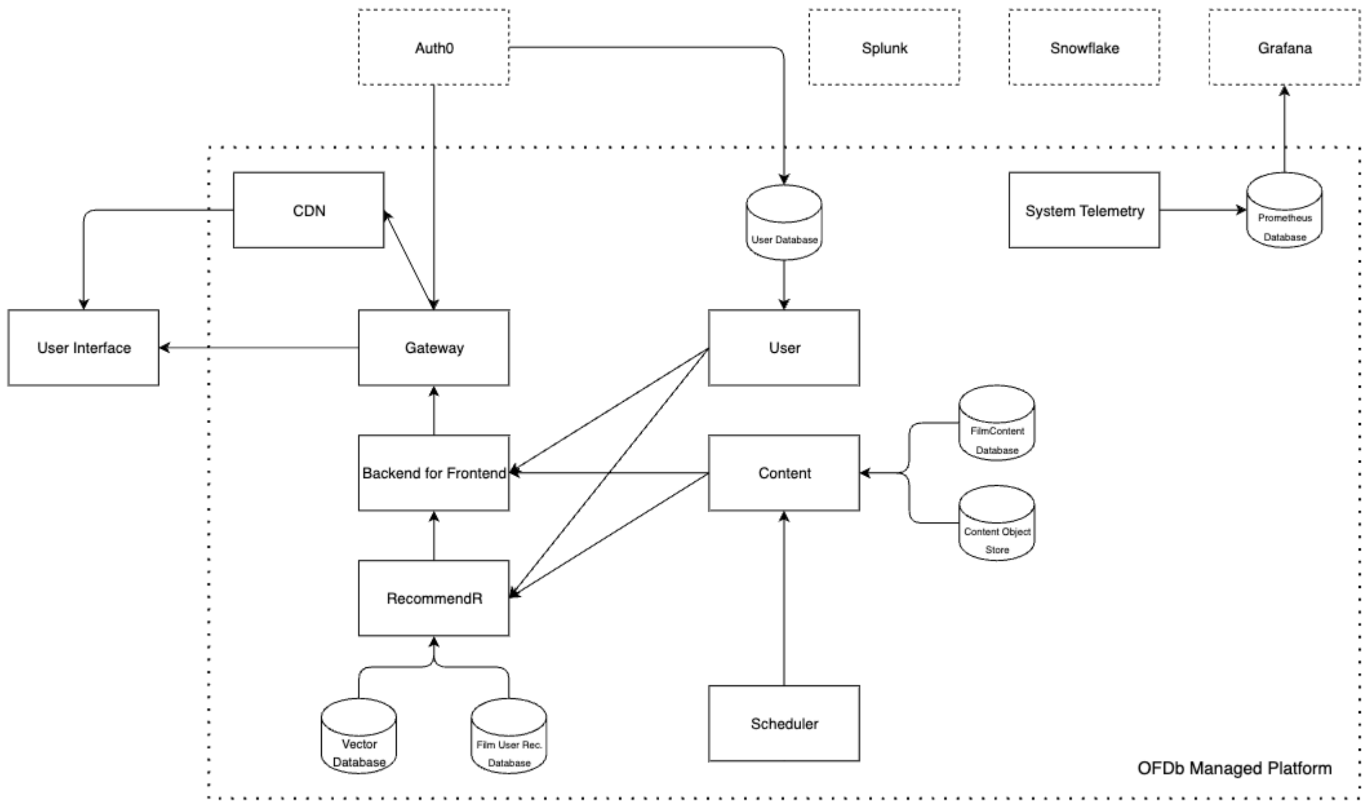


Figure 1: A system architecture diagram showing all high-level components of the OFDb platform and how they interact.

B.2 System Context Diagram

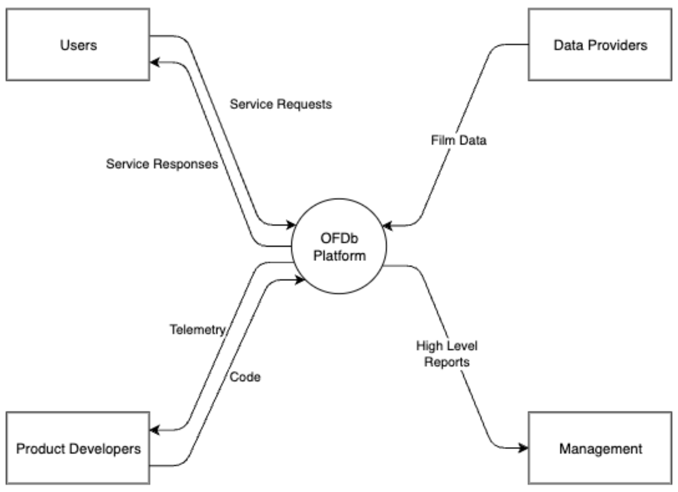


Figure 2: A system context diagram showing OFDb platform and the entities it interacts with.

C Data Models

C.1 Databases

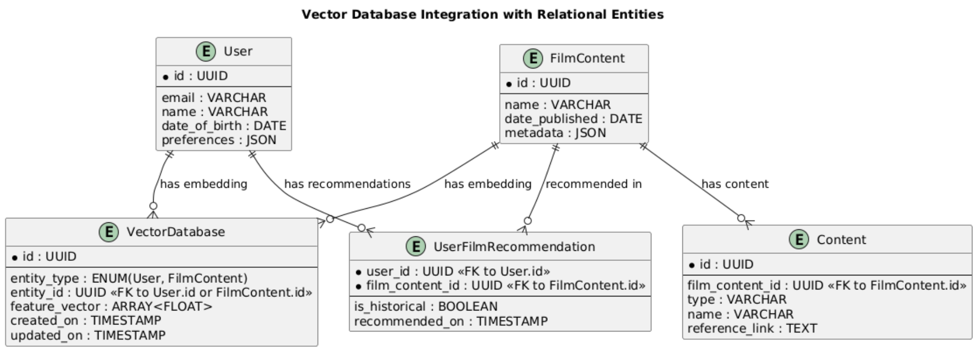
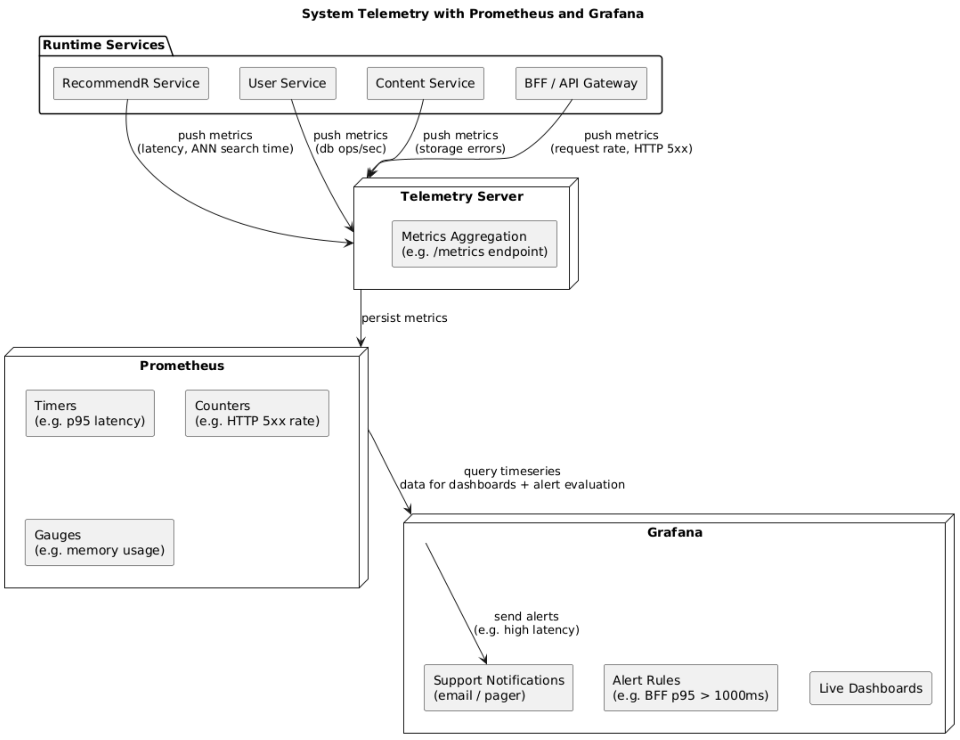


Figure 3: The normal form relational database tables of User, FilmContent, UserFilmRecommendation and Content, and how they interact with the vector database.

C.2 System Telemetry



System Telemetry with Prometheus and Grafana
Runtime Services
RecommendR Service
User Service
Content Service
BFF / API Gateway
push metrics (latency, ANN search time)
push metrics (db ops/sec)
push metrics (storage errors)
push metrics (request rate, HTTP 5xx)
Telemetry Server
Metrics Aggregation (e.g. /metrics endpoint)
persist metrics
Prometheus
Timers (e.g. p95 latency)
Counters (e.g. HTTP 5xx rate)
Gauges (e.g. memory usage)
query timeseries data for dashboards + alert evaluation
Grafana
send alerts (e.g. high latency)
Support Notifications (email / pager)
Alert Rules (e.g. BFF p95 > 1000ms)
Live Dashboards

Figure 4: Shows the interaction between components of the OFDb system and the system telemetry subsystem.

C.3 User Telemetry

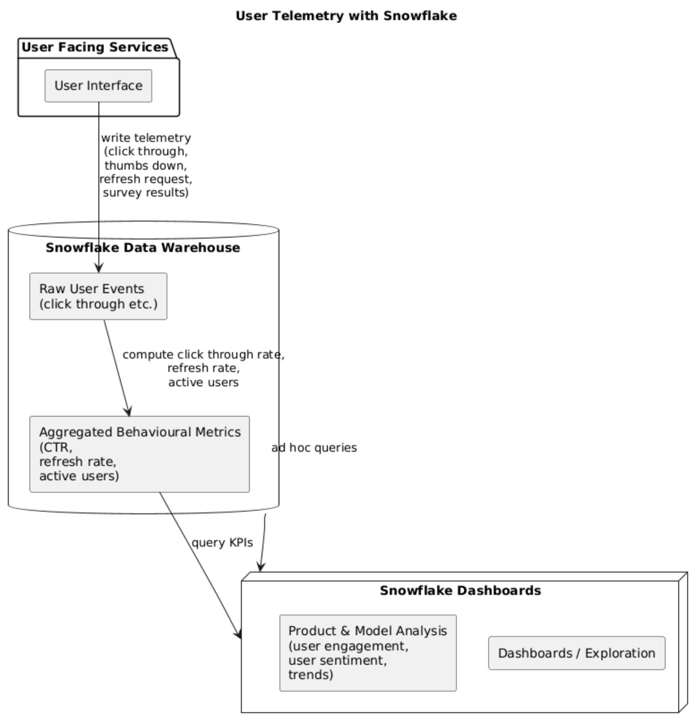

User Telemetry with Snowflake
User Facing Services
User Interface
write telemetry (click through, thumbs down, refresh request, survey results)
Snowflake Data Warehouse
Raw User Events (click through etc.)
compute click through rate, refresh rate, active users
Aggregated Behavioural Metrics (CTR, refresh rate, active users)
ad hoc queries
query KPIs
Snowflake Dashboards
Product & Model Analysis (user engagement, user sentiment, trends)
Dashboards / Exploration

Figure 5: Shows the interaction between the UI and the user telemetry subsystem.

D Fault Tree Analysis

D.1 Databases

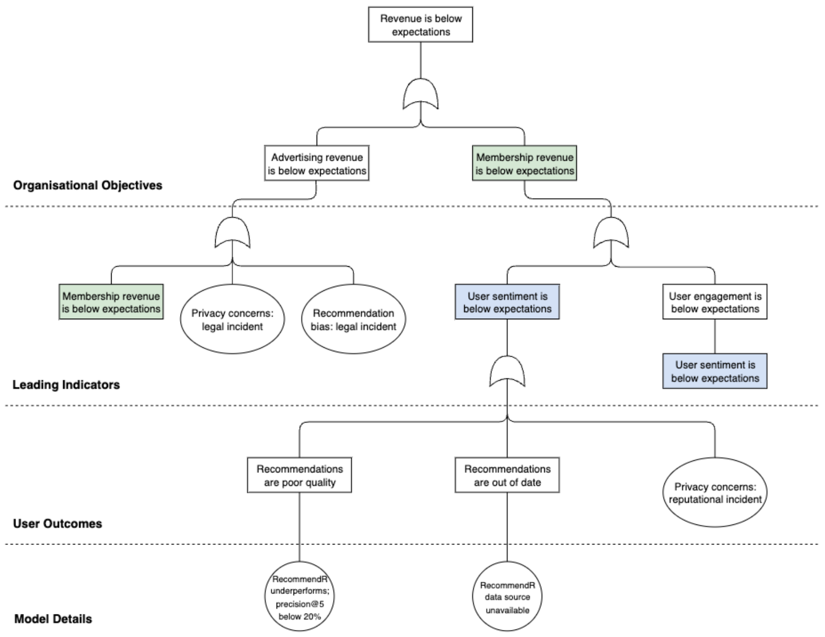

Revenue is below expectations
Advertising revenue is below expectations
Membership revenue is below expectations
Organisational Objectives
Membership revenue is below expectations
Privacy concerns: legal incident
Recommendation bias: legal incident
User sentiment is below expectations
User engagement is below expectations
User sentiment is below expectations
Leading Indicators
Recommendations are poor quality
Recommendations are out of date
Privacy concerns: reputational incident
User Outcomes
RecommendR underperforms; precision@5 below 20%
RecommendR data source unavailable
Model Details

Figure 6: A modified fault tree analysis diagram, detailing the relationships between the goals and their anticipated risks.

E Tables

Table 1: An enumeration of the key principles for software architecture design and how this system embodies them.

Software Architecture Principle	**Demonstrated By**	**Reasoning**
Modularity	Whole system	Every component of the system provides a distinct behaviour and interoperates to deliver the OFDb product. Each service (e.g., User, Content, RecommendR, Gateway, BFF) can be developed, maintained, tested and deployed independently.
Abstraction	BFF, Gateway, Vector database	The BFF hides backend service complexity from the UI by exposing a single, unified API. The Gateway abstracts authentication and routing. The vector database abstracts approximate nearest-neighbour search and similarity retrieval for RecommendR. Modular services further enforce abstraction by limiting what each component must know about others.
Separation of Concerns	Whole system	Each service is responsible for a single, well-defined concern. For example, the User service manages user data, the Content service manages film metadata and assets, and RecommendR generates recommendations. This avoids duplicated logic and reduces accidental coupling between unrelated behaviours.
Encapsulation	Individual services and internal data stores (UserDb, Content DB, vector DB)	Internal state is hidden behind service boundaries. For example, only the User service can directly modify the UserDb; other services must call its API. Similarly, RecommendR owns its ranking logic and vector retrieval process, which are not exposed directly to external callers. This enforces ownership and prevents uncontrolled access.
Flexibility and Extensibility	Content service, RecommendR pipeline, infrastructure-as-code	The Content service is designed to ingest new data sources without rewriting core logic. RecommendR's feature vectors combine hand-crafted features and learned embeddings, allowing new features to be added over time. Infrastructure-as-code and containerised deployment on Kubernetes/EKS allow evolution (new regions, new services) without fundamental redesign.
Scalability	Microservices running in containers on Kubernetes/EKS	Each service can be scaled horizontally by running additional instances behind a load balancer (e.g., scaling RecommendR separately from the User service). Vertical scaling remains possible by allocating more CPU/memory to specific pods. Autoscaling policies, driven by Prometheus/Grafana telemetry, enable responsive scaling while controlling cost.
Loose Coupling	Service APIs, message queue (Kafka), Gateway/BFF pattern	Services communicate via well-defined HTTP APIs and, where appropriate, asynchronous Kafka events, rather than shared state. This reduces interdependencies and allows internal changes without breaking contracts. The Gateway and BFF further decouple UI and backend evolution.
High Cohesion	Individual services	Each service groups closely related responsibilities. For example, all user preference and profile logic resides in the User service, while recommendation generation and filtering logic are contained within RecommendR. This improves maintainability and aligns ownership with team boundaries.

Table 2: Showing all the kinds of telemetry collected by the OFDb system.

Datapoint	**Telemetry Type**	**Storage**	**Published By**	**Purpose**
Container CPU Utilisation	System	Prometheus	All containers	Monitor instance compute utilisation using gauges; enables proactive auto-scaling.
Container Memory Utilisation	System	Prometheus	All containers	Monitor instance memory utilisation using gauges; enables proactive auto-scaling.
Request Latency	System	Prometheus	All containers	Monitor request latency with timers added to the Gateway server.
Recommendation Click Through	User	Snowflake	Web application	Recorded on recommendation click; aggregated and used as a proxy for precision@5.
Recommendation Focus	User	Snowflake	Web application	Record recommendation focus or visit time; aggregated and used as a measure of recommendation relevance.
Recommendation Thumbs Down	User	Snowflake	Web application	Recorded on recommendation "thumbs down"; aggregated and used as a proxy for the false positive rate.
Recommendation Refresh	User	Snowflake	Web application	Recorded on recommendation refresh; aggregated and used as a proxy for the false negative rate.
User Survey	User	Snowflake	Web application	Aggregated and used as a direct measure or proxy for user sentiment or perceived recommendation relevance.
Application Logs	System	Splunk	All containers	Persisted low-level system logs used to investigate and resolve production bugs and defects.