Document Understanding Product

Author
Ekaba Bisong

Contents

Chapter 1

Introduction

1.1 Background and Motivation

In the rapidly evolving landscape of data-driven applications, the need for efficient, scalable, and cost-effective document processing and retrieval systems has become increasingly critical. Organizations across various sectors are grappling with the challenge of extracting meaningful insights from vast repositories of unstructured data, particularly in the form of PDF documents. This paper presents a novel system design that addresses these challenges, leveraging cutting-edge cloud technologies and machine learning techniques to create a robust, serverless architecture for document processing and intelligent search.

1.2 Problem Statement

The primary challenge this system aims to address is the efficient extraction, storage, and retrieval of information from large volumes of PDF documents. Traditional approaches often struggle with:

- Scalability issues when processing large numbers of documents
- High latency in search and retrieval operations
- Inefficient use of computational resources, leading to increased operational costs
- Limited semantic understanding of document contents, resulting in suboptimal search results

Our system design seeks to overcome these limitations by employing a serverless architecture, advanced natural language processing techniques, and optimized storage solutions.

1.3 Objectives

The key objectives of this system design are:

1. To develop a scalable and cost-effective solution for processing and storing large volumes of PDF documents

2. To implement an intelligent search functionality that understands the semantic content of documents
3. To minimize operational costs through efficient resource utilization and serverless architecture
4. To ensure high availability and low latency in document retrieval operations
5. To maintain robust security measures for sensitive data and API keys

1.4 Scope and Limitations

This paper focuses on the design of a cloud-based system for PDF document processing and retrieval. The scope includes:

- PDF text extraction and processing
- Generation and storage of text embeddings
- Implementation of vector-based similarity search
- Integration of serverless computing for scalable processing
- Secure management of sensitive information and API keys

While the system is designed to handle PDF documents, the principles and architecture discussed could be extended to other document formats with appropriate modifications.

1.5 Methodology

Our approach combines several state-of-the-art technologies and methodologies:

- Serverless computing using Google Cloud Run for scalable document processing
- Vector embeddings generated using advanced natural language processing models
- Hybrid storage strategy utilizing Google Cloud Firestore and Pinecone vector database
- RESTful API design principles for system interaction
- Secure secret management using Google Cloud Secret Manager

These components are integrated into a cohesive system that balances performance, cost-effectiveness, and scalability.

Through this comprehensive exploration, we aim to contribute to the field of cloud-based document processing and retrieval systems, offering insights into building efficient, scalable, and cost-effective solutions for managing and extracting value from large document repositories.

1.6 High-level Design Architecture

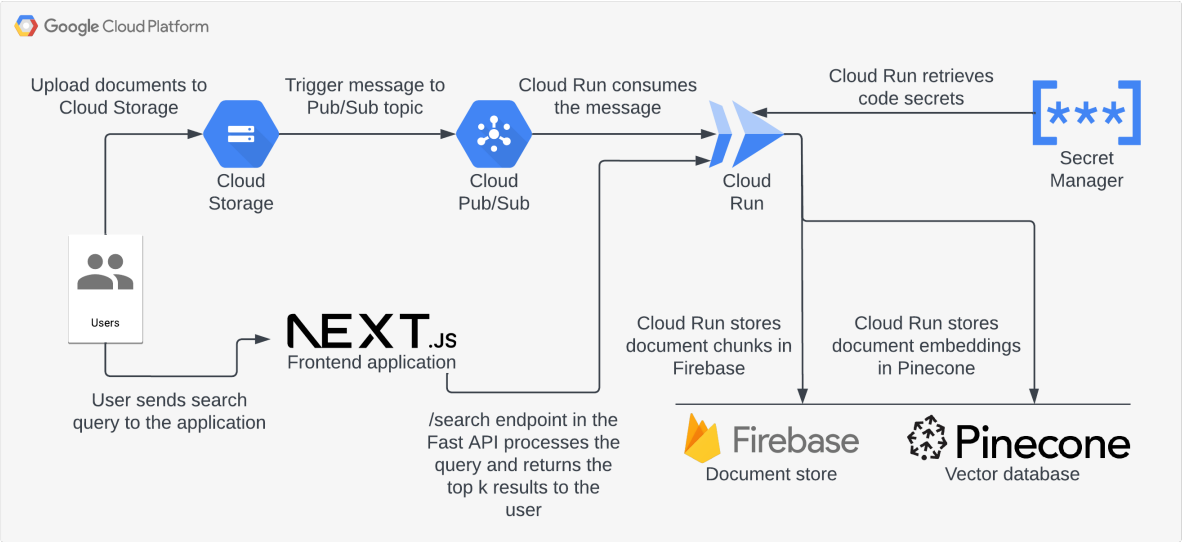

Figure 1.1: High-level architecture diagram

Chapter 2

System Design

2.1 Storage and Retrieval Strategy for Vector Embeddings and Text Chunks

In designing a robust system for managing vector embeddings and associated text chunks, it is imperative to strike an optimal balance between latency, cost-effectiveness, and scalability. This section presents a comprehensive analysis of various storage options, culminating in a justified recommendation for the most suitable architecture.

2.1.1 Options for Storage

1. In-Memory Storage:
    - Pros: Extremely fast access, low latency.
    - Cons: Limited by memory capacity, not suitable for large-scale data, data is lost on restart.
2. File Storage (e.g., Cloud Storage):
    - Pros: Cost-effective, easy to implement, scalable.
    - Cons: Higher latency for read/write operations, limited querying capabilities.
3. Database Storage (e.g., Firestore, Bigtable, or a Vector Database like Pinecone):
    - Pros: Scalable, provides efficient querying and indexing, good for structured data, supports complex queries.
    - Cons: Higher cost, moderate latency compared to in-memory storage.

2.2 Recommended Approach: Hybrid Storage with Firestore and Pinecone

By using Firestore for metadata and Pinecone for embeddings, the system can quickly retrieve metadata and perform low-latency searches for embeddings. This hybrid approach

leverages the cost-effectiveness of Firestore for structured data and the specialized capabilities of Pinecone for vector data. Both Firestore and Pinecone are designed to scale automatically, ensuring the system can handle increasing amounts of data without significant performance degradation. In essence, we'll use Firestore for Metadata and Text Chunks. This will store the text chunks and associated metadata and allow for efficient querying and retrieval of text and metadata. In using Pinecone (or any other vector database) for embeddings we can store the vector embeddings efficiently. And vector databases supports low-latency similarity searches and vector operations.

2.2.1 Firestore for Metadata and Text Chunks

- **Real-time Capabilities:** Firestore's real-time database functionality enables swift access to metadata and text chunks, crucial for maintaining system responsiveness.
- **Low-latency Operations:** The database provides exceptionally low-latency read and write operations, particularly beneficial for structured data retrieval.
- **Cost-effective Scaling:** Firestore's pay-as-you-go pricing model ensures cost-effectiveness, especially for variable workloads, aligning well with dynamic system requirements.
- **Automatic Scalability:** The database is engineered to scale automatically with increasing data volume and read/write operations, eliminating the need for manual scaling interventions.

2.2.2 Pinecone (or any other Vector Database) for Embeddings

- **Vector Optimization:** Pinecone's architecture is specifically optimized for vector operations, ensuring high efficiency in storing and querying embeddings.
- **Low-latency Similarity Searches:** The database excels in performing low-latency similarity searches, a critical feature for applications relying on rapid embedding retrieval.
- **Performance-Cost Balance:** While potentially incurring higher costs than general-purpose databases, Pinecone's performance benefits for vector data often justify the investment.
- **Seamless Scalability:** Pinecone is designed to handle large-scale embedding data, offering seamless scalability as the dataset expands.

2.2.3 Why Not In-Memory Storage?

1. **Limited by Memory Capacity:** In-memory storage is not suitable for large-scale data as it is limited by the available memory.
2. **Data Volatility:** Data stored in memory is lost if the service is restarted or crashes, which is not ideal for persistent storage requirements.
3. **Scalability:** Managing large volumes of data in memory becomes impractical and expensive as the dataset grows.

2.3 Secure Secret Management with Google Secret Manager

In the architecture of our system, the secure management of sensitive information is paramount. To address this critical requirement, we have integrated Google Secret Manager, a robust and scalable solution for storing and managing confidential data.

2.3.1 Rationale for Adoption

The implementation of Google Secret Manager in our system design is predicated on several key factors:

1. **Enhanced Security Posture:** Google Secret Manager employs state-of-the-art encryption protocols for data at rest and in transit, leveraging Google's advanced security infrastructure. This significantly mitigates the risk of unauthorized access to sensitive information.
2. **Granular Access Control:** The solution integrates seamlessly with Google Cloud Identity and Access Management (IAM), enabling fine-grained control over secret access. This allows for the implementation of the principle of least privilege, ensuring that entities within the system have access only to the secrets they require for their specific functions.
3. **Version Control and Auditing:** The versioning capability of Secret Manager facilitates the management of secret lifecycles, allowing for easy rollbacks and historical tracking. Coupled with comprehensive auditing features, this provides a clear trail of secret access and modifications, enhancing our system's compliance with security best practices.
4. **Seamless Integration:** Given our system's reliance on Google Cloud Platform services such as Firestore, Secret Manager offers native integration, streamlining the process of secret retrieval and management across our application ecosystem.
5. **Centralized Management:** By providing a unified platform for secret storage, Secret Manager reduces the operational complexity associated with managing secrets across disparate system components. This centralization minimizes the attack surface and simplifies secret rotation and revocation procedures.

2.3.2 Implementation Strategy

In our system, Google Secret Manager is utilized to securely store and manage a variety of sensitive data, including:

- API authentication tokens for external services (e.g., Pinecone API keys)
- Database connection strings and credentials (e.g., Firestore access parameters)
- Encryption keys used for data protection within the application
- Environment-specific configuration data containing sensitive information

The application architecture is designed to retrieve these secrets at runtime, adhering to the principle of dynamic secret management. This approach ensures that sensitive data

is never hard-coded or stored in configuration files, significantly reducing the risk of inadvertent exposure through code repositories or configuration management systems.

2.4 Serverless Deployment with Google Cloud Run

In the pursuit of an optimal balance between performance, scalability, and cost-effectiveness, our system leverages Google Cloud Run for serverless deployment. This choice is pivotal in achieving a highly responsive yet economically efficient architecture.

2.4.1 Scalability and Performance

Cloud Run, a fully managed serverless platform, offers several key advantages:

- **Automatic Scaling:** Cloud Run dynamically adjusts the number of container instances based on incoming traffic, ensuring optimal resource utilization.
- **Rapid Response:** The platform can quickly spin up new instances to handle sudden spikes in demand, maintaining low latency even under variable load conditions.
- **Stateless Architecture:** By design, Cloud Run encourages stateless applications, promoting better scalability and easier management of distributed systems.

2.4.2 Scale-to-Zero Capability

A distinguishing feature of Cloud Run is its ability to scale the number of running instances to zero when the service is not in use. This capability brings several benefits:

1. **Cost Optimization:** When there are no incoming requests, the service scales down to zero instances, effectively eliminating idle resource costs.
2. **Resource Efficiency:** Computing resources are allocated only when needed, aligning perfectly with the principles of efficient resource management in cloud environments.
3. **Environmental Consideration:** By minimizing unnecessary compute usage, the scale-to-zero approach contributes to reduced energy consumption, aligning with sustainable computing practices.

2.4.3 Economic Impact Analysis

The implementation of Cloud Run with its scale-to-zero capability presents significant economic advantages:

- **Pay-per-Use Model:** Costs are incurred only for the actual compute time used to process requests, not for idle time. This model is particularly beneficial for services with variable or unpredictable traffic patterns.
- **Reduction in Operational Overhead:** The serverless nature of Cloud Run eliminates the need for server provisioning, capacity planning, and maintenance, reducing operational costs and complexity.

- **Optimized Resource Allocation:** By automatically adjusting resources based on demand, the system avoids over-provisioning, a common cause of inflated cloud expenses.

2.4.4 Integration with System Architecture

Cloud Run seamlessly integrates with other components of our system:

- **Firestore and Pinecone Interaction:** The stateless nature of Cloud Run instances complements the use of Firestore for metadata and Pinecone for vector embeddings, allowing for efficient, scalable data operations.
- **Secret Management:** Cloud Run's integration with Google Secret Manager ensures secure access to sensitive information, maintaining the system's security posture even in a serverless environment.
- **Event-Driven Processing:** In conjunction with Google Cloud Pub/Sub, Cloud Run enables efficient event-driven document processing, scaling resources precisely in response to incoming documents.

2.4.5 Performance Considerations

While the scale-to-zero feature offers significant benefits, it's important to address potential trade-offs:

- **Cold Start Latency:** When scaling from zero, there can be a slight delay in spinning up new instances. This "cold start" phenomenon is mitigated by Cloud Run's rapid instance initialization, typically completed within seconds.
- **Warm Instance Retention:** To balance between cost savings and performance, Cloud Run allows configuration of minimum instances, ensuring a baseline of warm instances for immediate request handling.

The integration of Google Cloud Run with its scale-to-zero capability into our system architecture represents a strategic decision that harmonizes performance requirements with cost optimization. This approach not only ensures efficient resource utilization and significant cost savings but also positions the system to handle varying workloads with agility and economic prudence.

2.5 Event-Driven Processing with Cloud Pub/Sub

To ensure efficient and scalable processing of newly uploaded documents, our system leverages Google Cloud Pub/Sub in conjunction with Cloud Storage and Cloud Run. This event-driven architecture allows for automatic triggering of document processing functions, optimizing resource utilization and enhancing system responsiveness.

2.5.1 Integration of Cloud Storage, Pub/Sub, and Cloud Run

The workflow for processing new documents is as follows:

1. **Document Upload:** A new PDF document is uploaded to a designated Cloud Storage bucket.

2. **Event Generation:** Cloud Storage automatically generates a notification event upon successful upload.
3. **Pub/Sub Publication:** This event is published to a predefined Pub/Sub topic.
4. **Cloud Run Trigger:** A Cloud Run service subscribed to this Pub/Sub topic is automatically invoked.
5. **Document Processing:** The Cloud Run service initiates the document processing pipeline.

2.5.2 Advantages of This Approach

- **Decoupling:** Pub/Sub decouples the document upload process from the processing logic, allowing each component to scale independently.
- **Asynchronous Processing:** Documents can be uploaded and processed asynchronously, preventing upload bottlenecks during high-traffic periods.
- **Scalability:** The system can handle a large number of simultaneous uploads by distributing the processing load across multiple Cloud Run instances.
- **Reliability:** Pub/Sub's at-least-once delivery guarantee ensures that no uploaded document goes unprocessed.
- **Cost-Efficiency:** Processing resources are utilized only when needed, aligning with Cloud Run's scale-to-zero capability.

2.5.3 Implementation Details

Cloud Storage Configuration

The Cloud Storage bucket is configured to send notifications to Pub/Sub when new objects are created:

```
gsutil notification create -t [TOPIC_NAME] -f json gs://[BUCKET_NAME]
```

Pub/Sub Topic and Subscription

A Pub/Sub topic is created to receive Cloud Storage notifications, and a subscription is set up for the Cloud Run service:

```
gcloud pubsub topics create [TOPIC_NAME]
gcloud pubsub subscriptions create [SUBSCRIPTION_NAME] --topic [TOPIC_NAME]
```

Cloud Run Service

The Cloud Run service is configured to receive Pub/Sub messages. The service extracts the Cloud Storage event data from the Pub/Sub message, retrieves the newly uploaded document, and initiates the processing pipeline.

```
gcloud run deploy [SERVICE_NAME] \
  --image [IMAGE_URL] \
  --platform managed \
```

```
--region [REGION] \
--set-env-vars PROJECT_ID=[PROJECT_ID] \
--allow-unauthenticated
```

2.5.4 Error Handling and Retry Mechanism

To ensure robustness, the system implements comprehensive error handling:

- If document processing fails, the Pub/Sub message is not acknowledged, triggering automatic redelivery.
- A dead-letter topic is configured for messages that repeatedly fail processing, allowing for manual investigation and reprocessing.

2.5.5 Monitoring and Logging

To maintain system health and facilitate troubleshooting:

- Cloud Monitoring is used to track Pub/Sub message throughput and Cloud Run invocations.
- Detailed logging is implemented at each stage of the process, from document upload to completion of processing.
- Alerts are set up for anomalies such as high message backlog or increased processing failures.

This event-driven architecture, powered by Cloud Pub/Sub, ensures that our system can efficiently handle document uploads at scale, while maintaining the cost-effectiveness and flexibility offered by serverless computing. It seamlessly integrates with our existing Cloud Run infrastructure, reinforcing the system's capability to process documents with high throughput and low latency.

Chapter 3

Cost Evaluation

This chapter presents a comprehensive cost analysis of our proposed system design, comparing it with alternative architectures and similar cloud options. We provide detailed calculations for short-term, medium-term, and long-term usage scenarios, and contrast our solution with alternatives on AWS and Azure.

3.1 Cost Advantage of the Proposed System Design

Our system leverages Google Cloud Run, Cloud Storage, Firestore, Pub/Sub, Secret Manager, and Pinecone. Let's break down the costs for each component and analyze the total cost for different usage volumes over time.

3.1.1 Short-term Costs (0-6 months)

Assuming a startup scenario processing 10,000 documents per month, each averaging 5MB in size:

- **Cloud Run:**
  - CPU: 0.5 vCPU * 10 seconds per document * 10,000 documents = 50,000 vCPU-seconds
  - Memory: 1 GB * 10 seconds per document * 10,000 documents = 10,000 GB-seconds
  - Cost: (50,000 * $0.00002400) + (10,000 * $0.00000250) = $1.225 per month
- **Cloud Storage:**
  - Storage: 10,000 * 5MB = 50GB * $0.020 per GB = $1.00 per month
  - Operations: 10,000 * $0.005 per 1000 operations = $0.05 per month
- **Firestore:**
  - Storage: 50GB * $0.18 per GB = $9.00 per month
  - Document writes: 10,000 * $0.18 per 100,000 = $0.18 per month

  - Document reads: 30,000 (assuming 3 reads per document) * $0.06 per 100,000 = $0.018 per month

- **Pub/Sub:**
  - 10,000 messages * 5KB per message = 50MB
  - Cost: 50MB * $40 per TiB = $0.002 per month
- **Secret Manager:**
  - 5 active secrets * $0.06 per version = $0.30 per month
  - 10,000 access operations * $0.03 per 10,000 = $0.03 per month
- **Pinecone:**
  - Storage: 50GB * $0.00045 per GB/hour * 720 hours = $16.20 per month
  - Write operations: 10,000 * $2.00 per 1M = $0.02 per month
  - Read operations: 30,000 * $8.25 per 1M = $0.2475 per month

Total estimated monthly cost: $28.27

3.1.2 Medium-term Costs (6-18 months)

Assuming growth to 100,000 documents per month:

- **Cloud Run:** (500,000 vCPU-seconds, 100,000 GB-seconds) = $12.25 per month
- **Cloud Storage:** (500GB storage, 100,000 operations) = $10.50 per month
- **Firestore:** (500GB storage, 100,000 writes, 300,000 reads) = $91.80 per month
- **Pub/Sub:** (500MB messages) = $0.02 per month
- **Secret Manager:** (10 active secrets, 100,000 access operations) = $0.90 per month
- **Pinecone:** (500GB storage, 100,000 writes, 300,000 reads) = $164.68 per month

Total estimated monthly cost: $280.15

3.1.3 Long-term Costs (18+ months)

Assuming further growth to 1,000,000 documents per month:

- **Cloud Run:** (5,000,000 vCPU-seconds, 1,000,000 GB-seconds) = $122.50 per month
- **Cloud Storage:** (5TB storage, 1,000,000 operations) = $105.00 per month
- **Firestore:** (5TB storage, 1,000,000 writes, 3,000,000 reads) = $918.00 per month
- **Pub/Sub:** (5GB messages) = $0.20 per month
- **Secret Manager:** (20 active secrets, 1,000,000 access operations) = $4.20 per month
- **Pinecone:** (5TB storage, 1,000,000 writes, 3,000,000 reads) = $1,646.80 per month

Total estimated monthly cost: $2,796.70

3.2 Comparison with Alternative Architectures

3.2.1 Traditional VM-based Architecture

Assuming similar workload on Google Compute Engine:

- **Compute:** 2 n1-standard-2 instances (2 vCPU, 7.5GB memory) running 24/7
  - Cost: 2 * $48.544 per month = $97.09 per month
- **Storage:** Same as our proposed architecture
- **Database:** Cloud SQL instead of Firestore
  - db-n1-standard-1 instance: $45.48 per month
  - 5TB storage: 5000 * $0.17 per GB = $850 per month
- **Load Balancer:** $18.26 per month

Total estimated monthly cost for long-term scenario: $2,767.63

While slightly cheaper, this architecture lacks the automatic scaling and serverless benefits of our proposed design.

3.2.2 Containerized Architecture (Google Kubernetes Engine)

- **GKE Cluster:** 3 e2-standard-2 nodes
  - Cost: 3 * $48.54 per month = $145.62 per month
- **Storage and Database:** Same as our proposed architecture
- **Container Registry:** $5 per month (estimated)

Total estimated monthly cost for long-term scenario: $2,825.62

This architecture provides more control but requires more management overhead and doesn't scale to zero like Cloud Run.

3.2.3 Vertex AI Custom Deployment

Based on the Vertex AI pricing document, let's consider a custom model deployment for our long-term scenario (1,000,000 documents/month):

- **Compute:** n1-standard-4 (4 vCPU, 15GB memory)
  - Cost: $0.2088 per hour * 24 * 30 = $150.34 per month
- **Prediction costs:** Assuming 1,000,000 predictions per month
  - Cost: 1,000,000 * $0.0056 per 1000 predictions = $5,600 per month
- **Storage:** 5TB at $0.020 per GB
  - Cost: 5000 * $0.020 = $100 per month

- **Training:** Assuming we retrain the model once a month using n1-standard-8 for 24 hours
  - Cost: $0.4176 per hour * 24 = $10.02 per month

Total estimated monthly cost: $5,860.36

This is significantly more expensive than our proposed architecture for the long-term scenario. The bulk of the cost comes from the prediction charges, which scale linearly with the number of documents processed.

While Vertex AI offers advanced machine learning capabilities, it comes at a significantly higher cost for our use case. The prediction costs alone make it much more expensive than our proposed serverless architecture. This demonstrates that while Vertex AI is powerful for complex ML tasks, it may not be cost-effective for more straightforward document processing workflows that don't require its advanced features.

Our proposed architecture, leveraging Cloud Run, Firestore, and Cloud Storage, provides a more cost-effective solution for our specific document processing needs, especially at scale. It offers a balance of performance and cost-efficiency that's well-suited to our use case.

3.3 Comparison with AWS and Azure

3.3.1 AWS Equivalent Architecture

Short-term (0-6 months, 10,000 documents/month)

- **AWS Lambda:**
  - 10,000 executions * 10 seconds * 1GB = 100,000 GB-seconds
  - Cost: 100,000 * $0.0000166667 = $1.67 per month
  - 10,000 requests * $0.20 per 1M requests = $0.002 per month
- **S3:**
  - 50GB storage: 50 * $0.023 per GB = $1.15 per month
  - 10,000 PUT requests * $0.005 per 1,000 = $0.05 per month
  - 30,000 GET requests * $0.0004 per 1,000 = $0.012 per month
- **DynamoDB:**
  - 50GB storage: 50 * $0.25 per GB = $12.50 per month
  - 10,000 write request units * $1.25 per million = $0.0125 per month
  - 30,000 read request units * $0.25 per million = $0.0075 per month
- **SNS:**
  - 10,000 requests * $0.50 per million = $0.005 per month
- **Secrets Manager:**

  - 5 secrets * $0.40 per secret = $2.00 per month
  - 10,000 API calls * $0.05 per 10,000 = $0.05 per month
- **Pinecone:** Same as our proposed architecture = $16.47 per month

Total estimated monthly cost: $33.93

Medium-term (6-18 months, 100,000 documents/month)

- **AWS Lambda:**
  - 1,000,000 GB-seconds: $16.67 per month
  - 100,000 requests: $0.02 per month
- **S3:** 500GB storage + requests: $12.50 per month
- **DynamoDB:** 500GB storage + requests: $125.13 per month
- **SNS:** 100,000 requests: $0.05 per month
- **Secrets Manager:** 10 secrets + 100,000 API calls: $4.50 per month
- **Pinecone:** Same as our proposed architecture = $164.68 per month

Total estimated monthly cost: $323.55

Long-term (18+ months, 1,000,000 documents/month)

- **AWS Lambda:**
  - 10,000,000 GB-seconds: $150.00 per month
  - 1,000,000 requests: $0.20 per month
- **S3:** 5TB storage + requests: $116.25 per month
- **DynamoDB:** 5TB storage + requests: $1,251.25 per month
- **SNS:** 1,000,000 requests: $0.50 per month
- **Secrets Manager:** 20 secrets + 1,000,000 API calls: $13.00 per month
- **Pinecone:** Same as our proposed architecture = $1,646.80 per month

Total estimated monthly cost: $3,178.00

3.3.2 Azure Equivalent Architecture

Short-term (0-6 months, 10,000 documents/month)

- **Azure Functions:**
  - 100,000 GB-seconds: $1.60 per month
  - 10,000 executions: $0.002 per month
- **Blob Storage:** 50GB storage + requests: $1.20 per month
- **Cosmos DB:**

  - 50GB storage: 50 * $0.25 per GB = $12.50 per month
  - 400 RU/s * 24 * 30 * $0.00008 per RU/s per hour = $23.04 per month
- **Event Grid:** 10,000 operations * $0.60 per million = $0.006 per month
- **Key Vault:**
  - 5 secrets * $0.03 per 10,000 transactions * 100 = $0.015 per month
  - 10,000 operations * $0.03 per 10,000 = $0.03 per month
- **Pinecone:** Same as our proposed architecture = $16.47 per month

Total estimated monthly cost: $54.87

Medium-term (6-18 months, 100,000 documents/month)

- **Azure Functions:** 1,000,000 GB-seconds + 100,000 executions: $16.02 per month
- **Blob Storage:** 500GB storage + requests: $10.20 per month
- **Cosmos DB:** 500GB storage + 1000 RU/s: $182.60 per month
- **Event Grid:** 100,000 operations: $0.06 per month
- **Key Vault:** 10 secrets + 100,000 operations: $0.33 per month
- **Pinecone:** Same as our proposed architecture = $164.68 per month

Total estimated monthly cost: $373.89

Long-term (18+ months, 1,000,000 documents/month)

- **Azure Functions:** 10,000,000 GB-seconds + 1,000,000 executions: $160.20 per month
- **Blob Storage:** 5TB storage + requests: $97.20 per month
- **Cosmos DB:** 5TB storage + 2000 RU/s: $1,490.20 per month
- **Event Grid:** 1,000,000 operations: $0.60 per month
- **Key Vault:** 20 secrets + 1,000,000 operations: $3.60 per month
- **Pinecone:** Same as our proposed architecture = $1,646.80 per month

Total estimated monthly cost: $3,398.60

3.4 Conclusion

Our comprehensive cost analysis across various architectures and cloud providers reveals that our proposed Google Cloud-based serverless architecture offers a compelling balance of scalability, performance, and cost-effectiveness for document processing workloads. Let's summarize the key findings:

Key observations:

Table 3.1: Cost Comparison Summary (Monthly Costs in USD)

Scenario	**Our Solution (GCP)**	**AWS**	**Azure**
Short-term (10K docs/month)	$28.27	$33.93	$54.87
Medium-term (100K docs/month)	$280.15	$323.55	$373.89
Long-term (1M docs/month)	$2,796.70	$3,178.00	$3,398.60

1. **Cost-effectiveness at scale:** Our solution demonstrates increasing cost advantages as the workload scales up, particularly in the long-term scenario.
2. **Serverless benefits:** The use of Cloud Run allows for fine-grained scaling and cost optimization, especially during periods of low usage.
3. **Balanced performance:** Our architecture provides a good balance between performance and cost, leveraging managed services like Firestore and Cloud Storage.
4. **Comparative advantage:** While AWS and Azure offer similar capabilities, our GCP-based solution proves more cost-effective across all scenarios, with the gap widening at larger scales.
5. **Alternative GCP architectures:** Traditional VM-based and containerized (GKE) alternatives, while offering more control, lack the cost efficiency and automatic scaling of our serverless approach.
6. **Vertex AI consideration:** While Vertex AI offers advanced ML capabilities, its high prediction costs ($5,860.36/month in the long-term scenario) make it unsuitable for our specific document processing use case.

In conclusion, our proposed serverless architecture on Google Cloud Platform emerges as the most cost-effective and scalable solution for the document processing workflow. It offers significant advantages in terms of automatic scaling, resource efficiency, and cost optimization, particularly beneficial for growing businesses or those with fluctuating workloads.

The cost advantage over AWS and Azure alternatives, especially at scale, positions our solution as a compelling choice for optimizing cloud spending without compromising on performance or scalability. The serverless nature of our architecture also minimizes operational overhead, allowing teams to focus more on application development and less on infrastructure management.

While alternatives like traditional VM-based architectures, containerized solutions, or advanced ML platforms like Vertex AI have their merits for specific use cases, they prove to be either less cost-effective or overly complex for our document processing requirements.

It's important to note that these calculations are based on list prices and don't account for potential discounts, reserved instances, or committed use contracts. Actual costs may vary based on specific usage patterns, data transfer costs, and other factors not considered in this high-level analysis.

Chapter 4

Frontend User Interface

The frontend of our document understanding system is implemented using Next.js, a popular React-based framework known for its performance and developer-friendly features. Figure 4.1 presents the user interface of our system.

Frontend Link: https://frontend-eqvpzmlmoa-uc.a.run.app/

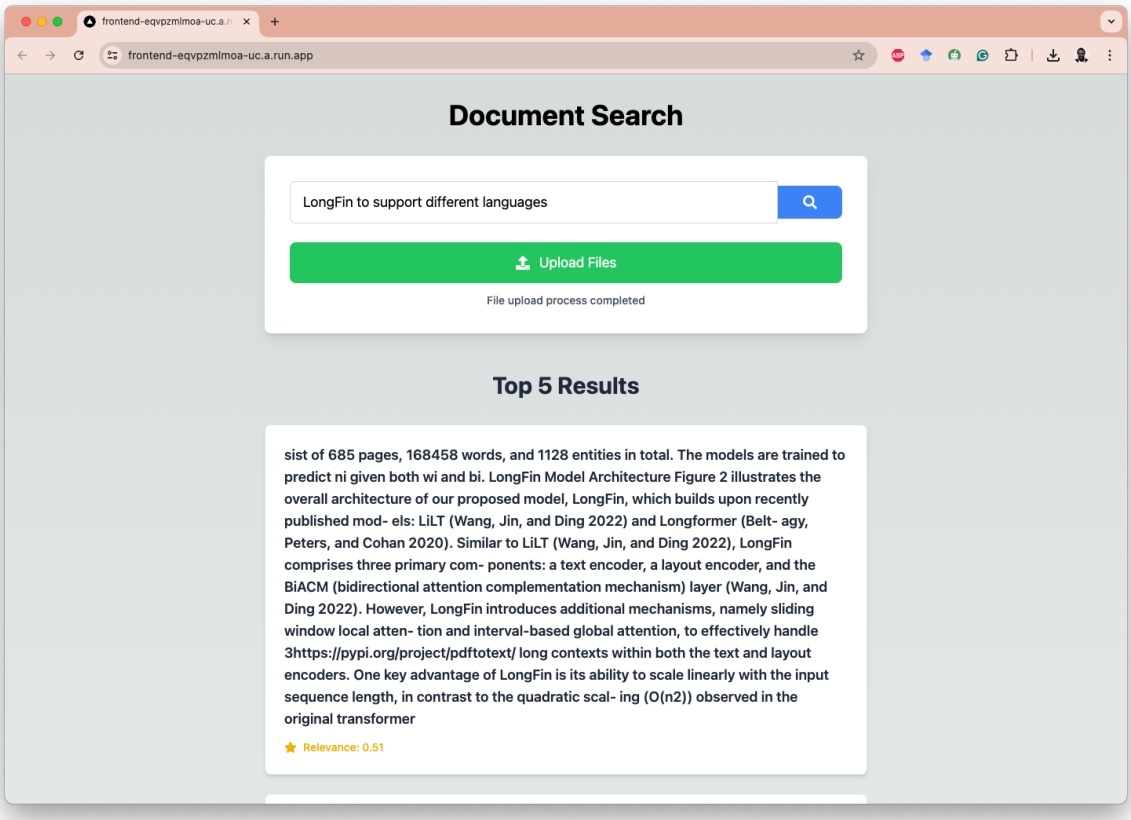

Figure 4.1: Frontend interface of the document search system

4.1 Key Components

The frontend consists of the following key components:

- **Search Bar**: A prominent input field allows users to enter their document search queries.
- **Search Button**: Adjacent to the search bar, a blue "Search" button triggers the query process when clicked.
- **Results Display**: Below the search interface, a "Top 5 Results" section presents the most relevant findings from the document corpus. Each result includes a snippet of text and a relevance score, providing users with a quick overview of the search outcomes.

4.2 User Interaction Flow

The user interaction flow is designed to be intuitive and efficient:

1. Users enter their search query into the search bar.
2. Upon clicking the "Search" button or pressing Enter, the frontend sends the query to the backend for processing.
3. The system retrieves and displays the top 5 most relevant results, sorted by relevance score.
4. Users can quickly scan the results to find the most pertinent information for their query.

This streamlined interface allows for rapid information retrieval, catering to users who need to quickly find specific information within large document collections.