National College of Ireland

Benchmarking AWS Bedrock Generative AI Models with RAG for Analyzing Fraudulent Transactions

MSc Research Project
Cloud Computing

Hunaid Vekariya
Student ID: X23235951

School of Computing
National College of Ireland

Supervisor: Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Hunaid Vekariya
Student ID:	X23235951
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Aqeel Kazmi
Submission Due Date:	12/12/2024
Project Title:	Benchmarking AWS Bedrock Generative AI Models with RAG for Analyzing Fraudulent Transactions
Word Count:	6518
Page Count:	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	*Hunaid Vekariya*
Date:	28th January 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Benchmarking AWS Bedrock Generative AI Models with RAG for Analyzing Fraudulent Transactions

Hunaid Vekariya
X23235951

Abstract

The proliferation of AI and Large Language Models (LLMs) are continuously evolving, with new models developed and tuned at high rate. However, they still facing limitations in providing domain-specific responses, particularly when dealing with fraudulent transactional data and financial knowledge. This challenge can be addressed through retrieval-augmented generation (RAG) systems deployed on serverless opensearch platform, which integrate vector databases to enhance model performance. This research benchmarks two advanced models from aws bedrock viz, Titan G1 premier and Mistral AI large. Comparing their performance based on execution time, token generation efficiency, comprehensiveness, and their compatibility with serverless event-driven architecture as these models are note evaluated on analyzing fraud activities. By analyzing how AI models can detect and mitigate fraudulent transactions, this study highlights the potential of combining LLMs with RAG systems to improve fraud detection and prevention. The evaluation findings are presented where Titan model provides more accurate and faster results, offering valuable insights and paving the way for further research opportunities.

Keywords— Prompt Engineering, AWS Bedrock, GenAI, Amazon Titan, Mistral AI, Credit Card Transaction, Opensearch Serverless

1 Introduction

Financial institutions are constantly evolving paving new ways to look at data and learn from it constantly. Artificial intelligence has been powerful in consuming a lot of data publicly available which starts making limitation for the models. The influx of ChatGPT by OpenAI , Claude by Anthropic and Gemini from Google are thriving the user market Prabhune and Berndt (2024).

Improving LLMs to retrieve information better can be achieved by pre-training approaches such as **p-tuning** Hu et al. (2021), **prefix tuning** Liu et al. (2021), **prompt tuning** Lester et al. (2021) talked about in paper . which can be challenging for an organization as it takes high resources with high mathematical neural calculation which can be achieved by GPUs only.

The other way to improve data generation and leveraging genai models is efficiently using vector database in form of RAG NVIDIA (2024) where the dataset is added into a centralized data lake in chunked manner which further provides context to the query requested by the prompt to provide generated responses in context with the data points. Building on these capabilities, our research delves into the use of Retrieval-Augmented Generation (RAG) implementations, which provide specific contexts for analysis and

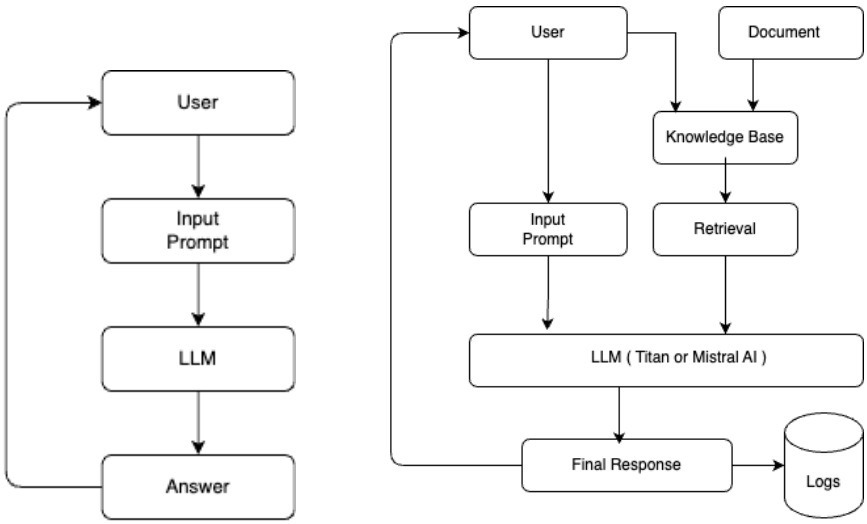

Figure 1: Flow - LLM and LLM with RAG

remediation strategies. This approach strengthens security systems to better prevent fraudulent transactions.

1.1 Motivation and Hypothesis

The lag between pre-tuning and fine-tuning can be optimized by using RAG while comparing embedded models like Titan and Mistral ultimately bridging the gap and need to further improve on whole model with huge dataset. The author of this article *Generative Artificial Intelligence: Prospects for Banking Industry* (n.d.) faces challenged as the financial banking data is critical and secured and cannot be used to train models for overall security risk. Our research can bridge this gap while keeping the data secured and encrypted in data lake and only indexing is deployed for vector database to retreive relevant information with sources. GenAI models are provided by all cloud vendors like GCP and Azure but AWS provides open source models directly available with unified API using boto3 sdk. This makes it advantageous over others as the comparison between open sources and proprietary models will be key highlight to our study.
This research deployed RAG with event-driven architecture using Bedrock-managed models to evaluate benchmarking between two models analyzing a payload of fraudulent transactions. The evaluation includes

- Metrics such as execution times
- Response comprehensiveness
- Tokens generated
- Cost-efficiency.

Despite their comparable strengths, this study aims to identify subtle performance differences under realistic workloads.

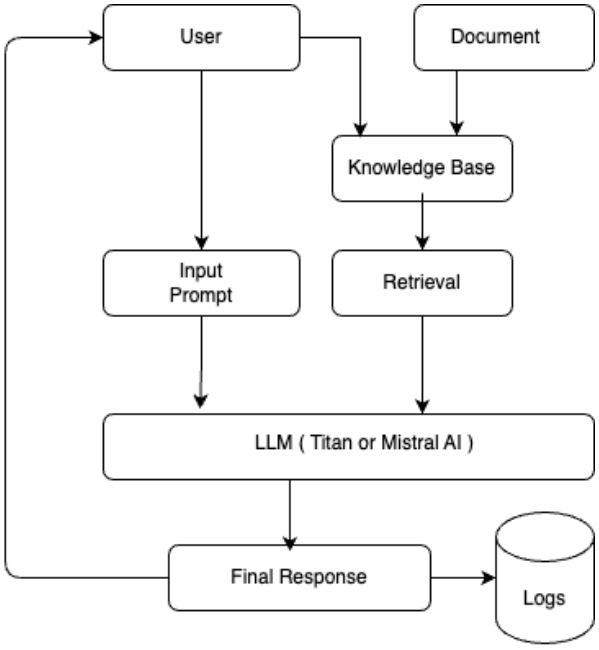

Figure 2: Enter Caption

Research Question
How effective are AWS Bedrock Generative AI models, such as Amazon Titan and Mistral, in analyzing fraudulent transactions through prompt engineering with RAG, and which model performs best in terms of execution time, tokens, and cost-efficiency?

The paper is structured into multiple sections. Section II reviews scientific literature, emphasizing benchmarking approaches and surveys highlighting areas where RAG can improve. Section III details the methodology, focusing on techniques and algorithms employed to build the experimental infrastructure. Section IV outlines the design specifications and describes the experimental setup for conducting the test cases. Finally, Section V presents the evaluation metrics, discusses the outcomes, and proposes directions for future research.

2 Related Work

Artificial intelligence has got immense support from the open-source community and tech giants such as Google, Meta, and Amazon. The work by Prabhune and Berndt (2024) explores the challenges insights of using Retrieval Augmented Generation(RAG) with large language models(LLMs), emphasizing efforts that enhance research into prompt engineering and fine-tuning. This Prabhune and Berndt (2024) paper discuses various methodologies for data retrieval in multi-NLP environments as implemented by Li et al. (2022). However, these methods are often not centralized as different data types requiere different models to create the vector store. This limitation is mitigated through the use of the AWS Bedrock knowledge base, which automates chunking and parsing processes.

2.1 Why RAG is Important?

RAG as highlighted by **?**, functions by combining traditional data retrieval methods with advanced generative model whcih utlizes architectures such as GPT-3 and BERT. When an API request is initiated, the GenAI model retrieves relevant information from the vector database before generating a response. This two-step approach ensures that outputs are not only accurate but also enriched with contextual and current information.

There are various RAG models and challenges associated with processing unstructured data, as discussed in Li et al. (2022), Zhao et al. (2024), and Yu et al. (2024). These challenges often arise due to the diverse nature of data types and the lack of uniformity in vector store creation. AWS Bedrock knowledge base addresses these issues by automating the chunking and parsing of data, which in return helps in improving efficiency and accuracy.

In the domain of fraud detection, GenAI uses RAG capabilities to deliver deeper insight. By cross examin transactional data with huge datasets, it identifies patterns, anomalies, and connections that might be missed by traditional models. This integration enhances the robustness of fraud detection systems, providing actionable insights and enabling more effective preventive measure

The article Singh (2024) highlights challenges in implementing RAG due to data security concerns. However, adopting robust approaches, such as storing data in encrypted storage systems, can mitigate these issues. This research addresses these concerns by proposing methods to enhance data protection without compromising functionality.

Enhanced Data Security and Integrity: Sensitive data remains confined within the controlled environment of the knowledge base, significantly reducing the risk of exposure or misuse.

Regulatory Compliance: Adhering to regulations like GDPR and other data protection laws becomes more manageable as sensitive information does not directly interact with external AI systems.

Improved Accuracy and Contextual Relevance: The RAG system ensures that GenAI models generate responses enriched with organizational context, leading to more precise and actionable insights.

This literature review underscores the importance of innovative frameworks that promote the ethical, secure, and efficient deployment of GenAI in financial systems. It emphasizes the critical role of RAG systems in addressing some of the industry's most pressing challenges, such as fraud detection, risk management, and compliance.

Enhancing RAG with Advanced Retrieval Techniques: The quality of responses generated by RAG systems can be significantly improved through advanced retrieval methods, such as:

Interactive Retrieval: Borgeaud et al. (2022) and Arora et al. (2023) proposes systems that refine queries interactively for better contextual relevance. Recursive Retrieval: Trivedi et al. (2023) and Kim et al. (2023) highlight the benefits of recursive approaches, which involve scanning documents and indexing data chunks to optimize external dataset re-ranking. Adaptive Retrieval: Jiang et al. (2023) introduces adaptive retrieval mechanisms, enabling systems to dynamically adjust retrieval strategies based on query context. This research incorporates recursive techniques for document scanning and indexing, which improve re-ranking of external datasets. These approaches also increase the output token limit for LLMs, resulting in more comprehensive and detailed responses.

2.2 The Role of GenAI in Fraud Detection, Risk Management, and Compliance in Financial Systems

Generative AI (GenAI) has begun transforming the finance and banking sectors by enabling advancements in fraud detection, risk management, and document summarization. According to a journal article published by the State Bank of India **?**, GenAI is being utilized to detect and prevent fraudulent activities, summarize extensive documents, and enhance risk assessment processes within financial institutions. These applications highlight GenAI's potential to optimize operations and decision-making in an industry that requires precision and reliability.

Enhanced Data Security and Integrity: Sensitive data remains within the controlled environment of the knowledge base, reducing the risk of exposure or misuse.

Regulatory Compliance: Adherence to GDPR **?** and other data protection laws becomes more manageable as sensitive information does not directly interact with external AI systems Hacker et al. (2023).

Improved Accuracy and Contextual Relevance: The RAG system ensures that the GenAI model's responses are informed by organizational context, leading to more precise and actionable insights.

2.3 Comparing GenAI Models

Comparing open-source models such as Mistral AI Large with proprietary ones like Amazon Titan Premier is essential to identify differences in analyzing prompts for fraud detection. For instance, one study Sallam et al. (2024) compared various LLM models, including Mistral AI Large, LLaMA, Meta, GPT-4, Gemini, and AWS Titan, in a Tic-Tac-Toe game simulation. GPT-4 achieved the highest success rate, while Titan was excluded due to the unavailability of JSON-formatted outputs—a limitation now addressed in the Titan Premier model.

—

2.4 The Role of Recursive and Adaptive Retrieval in RAG

The quality of RAG responses can be further improved through advanced retrieval techniques, such as interactive retrieval Borgeaud et al. (2022), retrieval on recursion Trivedi et al. (2023), and adaptive retrieval Jiang et al. (2023). Incorporating recursive approaches, such as document scanning and indexing data chunks, enhances the re-ranking of external datasets. These methods also increase output token limits for LLMs, allowing for more detailed and accurate responses.

—

2.5 Benchmarking GenAI Models

The volume and quality of data processed in GenAI models directly impact response accuracy and introduce potential biases, as highlighted by Bian et al. (2024) and Sallam et al. (2024). These biases can be mitigated by using private datasets specific to an organization, ensuring source reliability. Regular benchmarking of models in terms of response generation, response time, and compatibility with vector databases is critical for fully evaluating their efficiency and robustness.

—

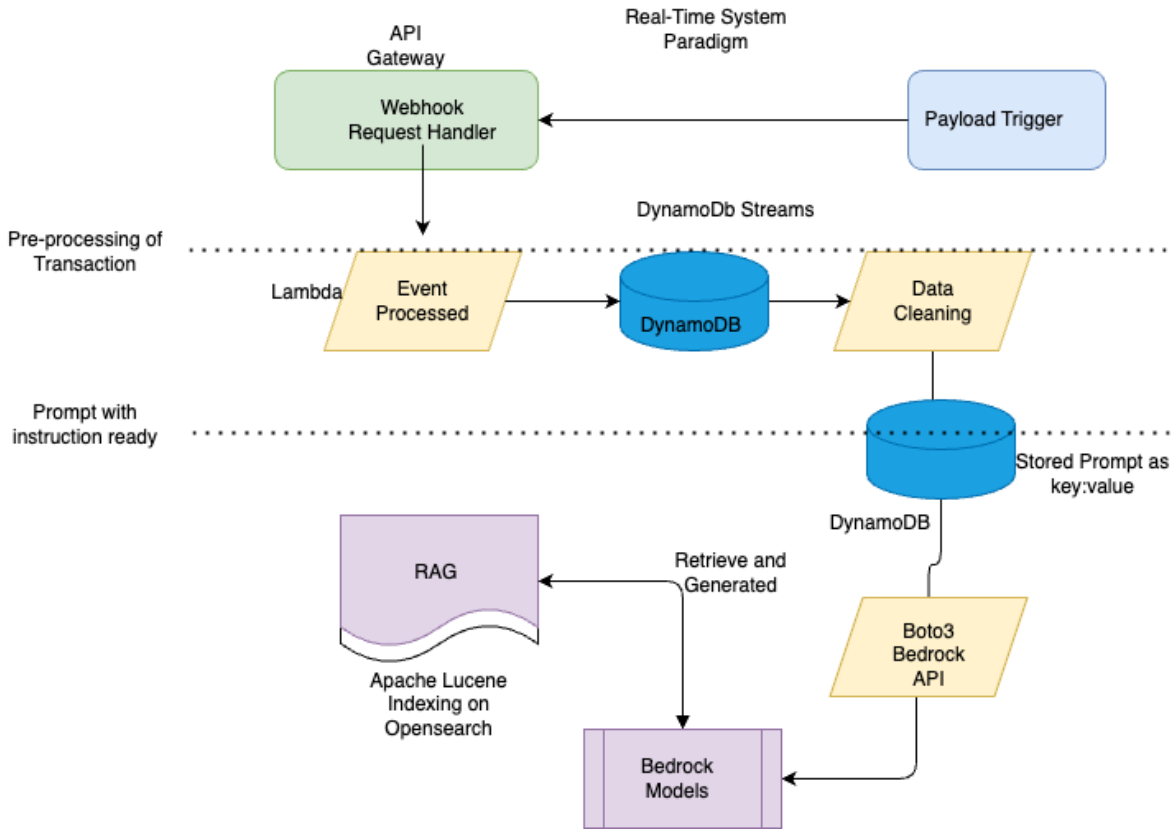

Figure 3: Event Driven Methods

2.6 Conclusion

Despite extensive research into integrating RAG, services like AWS Bedrock and OpenSearch Serverless present promising solutions to the challenges of running RAG with heterogeneous, unstructured data from diverse sources. Knowledge bases address many challenges by automating data chunking and parsing while ensuring compatibility with GenAI models. This capability makes them ideal for scenarios involving complex and varied data types, paving the way for more robust and scalable AI-driven systems.

3 Methodology

The hypothesis focuses on how good the genAI can analyze transactions and explore the the mitigation strategies using the data source we provided. The proposed system uses a cloud based architecture deployed on AWS to process the data streams in real-time in an event driven manner. Transactional payloads are simulated following the default schema from Stripe transactions payload but with synthetic data is taken from Kaggle and converted into json schema provided in artifacts[1]

The study employs a data-driven approach leveraging serverless technologies, including AWS Lambda, API Gateway, Serverless OpenSearch indexing, and AWS Bedrock. The architecture ensures scalability, efficiency, and seamless integration of real-time data pipelines.

[1] `https://www.kaggle.com/datasets/ealaxi/paysim1`

3.1 Quantitative Analysis:

This methodology uses synthetic transaction data sourced from Kaggle, containing flagged indicators which are further modified for failed transactions. The sourced data from 10s of location and urls provides the context and is stored in block storage with indexing provided by Apache Lucene running Opensearch indexing.

3.2 Serverless Models

Serverless triggers and rest api are used to stream payload from one function to another function. These methods solve the issue of unreliability and packet drops. Further serverless functions like lambda handles failure and process it again. The API endpoint is publicly accessible, allowing transaction payloads to be sent as webhooks. A request validator ensures the integrity and compliance of incoming payloads in the method request stage.

3.3 Functions and Text Summarization

Compute layer is used to run the logic before sending the prompt to bedrock agents like titan and mistral for text summarization and analysis. Validated payloads are sent to an AWS Lambda function. The function processes the data and stores the JSON response in a DynamoDB table, facilitating structured storage and retrieval for subsequent analysis. Models are pre-trained to provide text-summarization for the payload processed.

3.3.1 Prompt Engineering and AI models

Prompt engineering is relatively a new disclipline in the AI age. It not just about giving instructions to genai model to generate responses but needs engineering on top of it to provide precise elaborated prompts. Big prompts involve more tokens and every model provides a capacity for tokens to be consumed. It is very important to understand prompts in order to interact with the LLMs. **Amazon Titan Text G1** - Premier is one genAI model which provides the context window of 32k tokens and inference parameters for temperature and Top P as 0.7 and 0.9 respectively. Titan is propreitary to Amazon and cannot be self-deployed On the other hand, **Mistral AI large** provides 32k tokens for context window and is available to be self deployed which makes it open source.

3.4 Retrieval Augmented Generation

Pre-trained LLMs may not perform out of the box for every particular case so pretraining of the model is required. RAG is retrieval augmented Generation system can conceptualize the data provided and provide the responses based on that. The dimensions of data must be high about 1000 for it to work properly. Data chunking and parsing is further explained in the implementation part of this paper. RAG is composed of three components, retrieval, augmentation and generation. Upon the user query for the transaction, relevant content on transactional data is retrieve from external knowledge bases or other data sources based on the prompt query. In our case we retrieving the RAG from AWS Opensearch and 7 external URL sites in order to increase the dimensions of data. The retrieved information from Knowledge base is then appended to original user query creating

an augmented query to serve as the input to the foundational model. The foundational model then generated the response based on the augmented query. With this high-level flow, the use of rule based and semantic search is used where our RAG model is used. Rule based fetches the unstructured data like documents, emails etc. Semantic search fetches relevant documents based on their text embeddings.

3.4.1 Embedding

Embedding refers to transforming data like text, audio, images into numerical representation high dimensional vector space using machine learning algorithm which has been performed by AWS Knowledge base in bedrock service. For embedding models, factors such as max input size, latency, output embedding size, accuracy are crucial considerations. AWS provides two models for embedding Titan embedding and Mistral AI Large embedding models which have been used in this research to compare and contrast for further use. Two datasets points are used in this research, the one is synthetic data which is used for the prompt and is in json format. It includes the unique transactional id, location, flags, amounts, created date, updated date and other parameters. The other dataset is the data sourced into aws knowledge base which is taken from many renowned websites like Stripe radar, credit card frauds, fin tech etc. No machine learning training has been performed in this research. This research explores the capabilities of two Generative AI models, Amazon Titan Premier and Mistral AI Large, accessed through AWS Bedrock. AWS Bedrock provides a unified API for interaction, which is implemented using the boto3 SDK to ensure consistent and efficient communication with the models. The choice of these models was driven by their relevance to tasks requiring contextual understanding, summarization, and code generation.

3.4.2 Amazon Titan Premier

Amazon Titan Premier is a proprietary model developed by AWS. Known for its advanced text-generation capabilities, it is particularly adept at handling open-ended tasks, contextual understanding, and summarization. The model is exclusively hosted on AWS Bedrock, providing high availability and scalability but limited to the AWS ecosystem.

3.4.3 Mistral AI Large

Mistral AI Large, on the other hand, is an open-source model introduced by Mistral AI. It is optimized for high-performance tasks such as mathematical reasoning, text summarization, and long-context applications. With 123 billion parameters, the model supports single-node inference and offers the flexibility of deployment both within AWS Bedrock and on custom infrastructure, making it a versatile tool for experimentation. The experimental setup involved sending a series of predefined prompts to both models via the AWS Bedrock API. Responses were collected programmatically using Python scripts integrated with the boto3 SDK. Tasks were carefully designed to reflect real-world use cases, such as open-ended problem-solving, summarization, and code generation. The consistent use of AWS Bedrock as a hosting platform ensured uniformity in model deployment, minimizing infrastructure-related variations that could influence outcomes. The data generated through these interactions was prepared for analysis by standardizing response formats and ensuring comparability across models. This methodology provides a robust framework for evaluating the strengths and limitations of these models in subsequent sections.

3.5 Services Used in the Research Infrastructure

This research leverages a combination of AWS services to build an efficient, automated infrastructure for comparing Generative AI (GenAI) models. The automation of prompt handling accelerates the interaction and evaluation of these models, providing actionable insights. Below is a detailed description of the services utilized:

3.5.1 API Gateway

API Gateway acts as the entry point for processing payloads sent via HTTP POST requests. It forwards these requests to a designated Lambda function for further processing. Once the Lambda function processes the payload, it returns an HTTP 200 status code to confirm successful handling.

3.5.2 AWS Lambda

Lambda functions play a pivotal role in the automation pipeline. Three Lambda functions are deployed to:

- Pre-process incoming transaction data.
- Convert the processed data into structured prompts.
- Invoke the AWS SDK for Bedrock to generate responses from the knowledge base or models.

These serverless functions enable dynamic data handling and orchestration across the infrastructure.

3.5.3 Amazon DynamoDB

DynamoDB is employed as the primary storage layer for JSON-formatted key-value pairs. It supports DynamoDB Streams, which automatically trigger additional Lambda functions to handle new or updated data, ensuring real-time processing and integration with downstream services.

3.5.4 IAM (Identity and Access Management)

Properly defined IAM roles and policies are crucial for ensuring seamless communication between services. These roles provide the necessary permissions to avoid "Access Denied" errors while maintaining security best practices.

3.5.5 AWS Bedrock

AWS Bedrock serves as the core platform for interacting with GenAI models. It provides a unified API for model invocation, allowing prompts and instructions to be sent to models or Bedrock Agents. Additionally, Bedrock enables knowledge base integration, which can be further enriched using third-party datasets for improved contextual understanding.

3.5.6 OpenSearch Serverless

OpenSearch is used to index data points and manage a vector database. This facilitates Retrieval-Augmented Generation (RAG) by enabling the prompts to query the vector database for additional context. OpenSearch acts as a middleware, enhancing the accuracy and relevance of model outputs.

3.5.7 Amazon S3

S3 offers cost-efficient block storage for financial documents and other research artifacts. Its scalability ensures secure and reliable data storage throughout the study.

3.5.8 Amazon CloudWatch

CloudWatch logs are utilized to capture all responses generated by the infrastructure. This logging mechanism aids in debugging, performance monitoring, and comparative analysis of model outputs.

The methodology is most suitable here because it follows the best practice for serverless architecture and manages the resources very well. Running models on physical servers and creating a prompt engineering instructable program is out of scope for the scope

3.6 Evaluation Methedology

The research evaluates metrics and key performance indicators to find the effectiveness of RAG and compare the models responses with respect to compprehensiveness, execution time and tokens generated.

The techniques used experimental studies with observation on the responses generated. Simulations numbers are discussed in evaluation part.

4 Design Specification

4.1 Infrastructure Design

The **infrastructure diagram** in Fig. 2 illustrates an *event-driven architecture* designed to ensure **resiliency, redundancy, and reliability** across the system. Each service is seamlessly integrated with others through triggers and data streams, enabling efficient and scalable communication. This design leverages event-driven principles to optimize the flow of information while minimizing latency and bottlenecks.

4.2 Knowledge Base and Vector Storage

To support the vast amounts of data required for building and improving the knowledge base, an **OpenSearch Serverless cluster** is employed alongside an **S3 bucket**. This combination offers a cost-effective solution for managing terabytes of data, ensuring scalability and flexibility as the dataset dimensions grow over time. The serverless nature of both services accommodates dynamic scaling demands, providing a robust foundation for maintaining and querying the vector store.

4.3 Compute Power and Logic Execution

The computational logic is distributed across **multiple AWS Lambda functions**, which handle the processing tasks and execute SDKs to connect with the **AWS Bedrock service**. These Lambda functions are optimized to dynamically allocate compute power, ensuring efficient execution of tasks while maintaining low operational costs. The modular design of Lambda functions also facilitates easier maintenance and upgrades as the system evolves.

4.4 Data Flow and Response Management

The **flowchart** outlines the end-to-end flow of data and responses at each stage of the architecture. This includes data ingestion, processing, retrieval, and response generation, highlighting the interactions between various services and components. The design ensures that responses are both accurate and contextually relevant, leveraging the knowledge base for enhanced insights.

4.5 Key Features

- **Event-Driven Integration**: Services are interconnected via triggers and data streams, promoting real-time communication and adaptability to system changes.
- **Cost-Effective Scaling**: The combination of OpenSearch Serverless and S3 ensures scalability without incurring significant operational costs.
- **Modular Compute Power**: AWS Lambda functions provide a flexible and efficient mechanism to execute logic, making the system resilient and easily extensible.
- **Enhanced Knowledge Base**: The vector store grows dynamically, improving response accuracy over time through scalable and efficient data storage.

5 Implementation

Deployment of the event-driven architecture is carried out into multiple stages, from webhook hitting api gateway to processing it, converting it to prompt. The archiecture was based on the best practices scenarios from AWS, to ensure the systems remain reliable and resilient.

5.1 Core Architecture

AWS provides integrated services to connect api gateway, lambda function and dynanmodb. The architecture utilizes dynanomdb streams to trigger subsequent lambda functiin, enabling a fully event-driven workflow. The integration of bedrock using the boto3 software development kit completes the flow allowing intereations with genai models and knowledge base. The events dominate the triggering of the lambda functions and its fully automated. Webhook hits the api gateway which is directly integrated with lambda function which uses json body to upload the item to dynamodb as put_item operation in python. Dynamodb streams passes the inserted item to another lambda function which

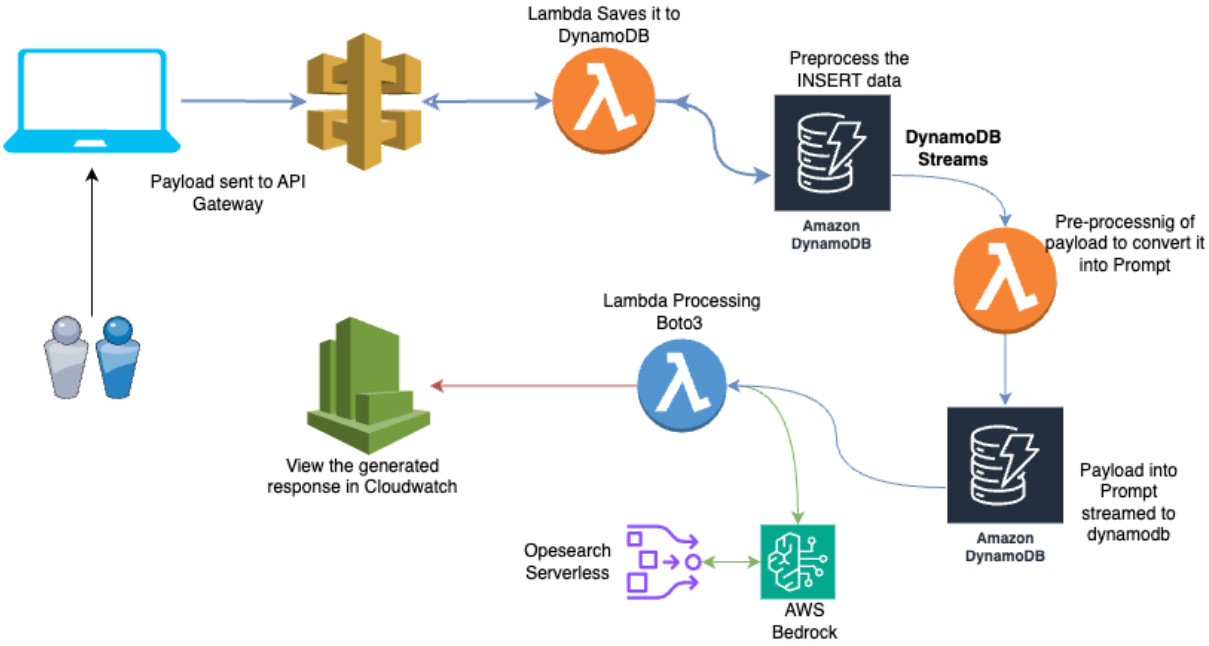

Figure 4: Infrastructure Design: Built on event driven architectural design to trigger function on new event, the cycle ends at logging the results into cloudwatch logs

pre-processes the data removes unwanted fields and returns the f strings into a prompt to different dynamodb table. Similar dynamodb table cannot be used as it would introduce infinite loop of events.

5.2 Implementing Knowledge Base - Retreive and Generate APIs

Knowledege base is created via aws bedrock console allowing multiple data sources from web crawler with default urlstrings and s3 bucket. Adding the s3 bucket ensures that all the data added further to the s3 bucket will be chunked and parsed using the knowledge base built in.

5.2.1 Standard Chunking

We have used default chunking which chunks the text in 300 tokens. Chunking provides the feature to view the sources once the resposne is recorded. Other chunking procedures are hierarchical and semantic but they are out of scope of this research.

5.3 RetrieveQuery

To query the knowledge base using the GenAI model, specific fields must be included in the request to ensure proper functionality. These fields include the input, which contains the query or data to be processed, and the sessionId, which helps maintain the sessions continuity for consistent interactions.

A retrievalQuery is sent in the request body as part of the query to the knowledge base along with the knowledgebaseID, which is included as a request parameter. The knowledgebaseID is critical as it ensure the query is directed to the correct knowledge

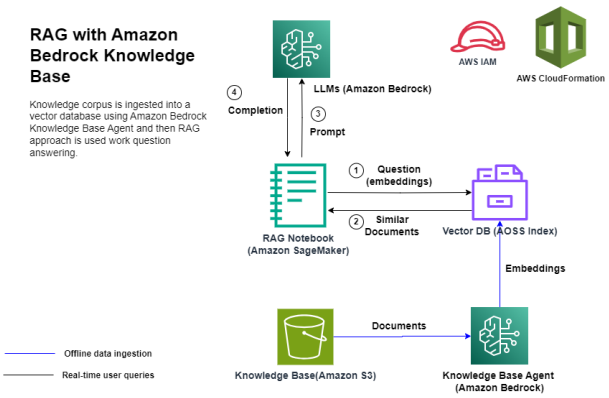

Figure 5: Knowledge Base Workflow

base.These components are required and must accompany the promt when interacting with the knowledge base.

This process occurs through the retrieve API which communicates with the knowledge base to fetch relevant information. Fig 4 descrives how this interaction works, showing how the API processes the request and retrieves the necessary data. This setup allows the GenAI model to leverage the knowledge base effectively, providing accurate and contextually relevant responses.

5.4 API Gateway Configuration

API Gateway receives webhooks and forwards them to a Lambda function. A resource must be created with a POST method to integrate Lambda. We are not using any key based authentication for the api gateway and the api developed is in REST API which provides better control on validating the body. The body should be validating with the schema provided in the artifacts with code.

The first lambda function to store the payload to dynamodb requires a put_item function inbuilt to python library. The second lambda function which pre-processes the payload and convert it to prompt requires more memory and execution time as many fields were dropped and only important fields like transactionId, flags, amount, location, merchantCategory and fraudscore were taken to create the prompt with intstructions to provide remidiation so these transactions can be further stopped from happening.

The third final lambda function implments the retrieve api to the bedrock knowledge base and bedrock gen ai models which does all the heavy lifting to udnerstand the prompt and retreive the context based responses from the provided knowledge base in the opensearch serverless

5.5 NoSQL and DynamoDB streams

The events are stored in a key:value pair in AWS dynamodb tables which allows fast nosql queries on the data.

1. The dynamodb uses the put_item operation to receive the json. The database consisted of three items

2. Tables - Multiple tables were created to keep the events new on every new insert

3. Items - The update to the table on every new payload and prompt added by lambda.

4. Atributes - The data inside every item is attributed as attribute. The data consisted of key:value pairs

5.5.1 DynamoDB Streams

Dynamodb streams play an important role in passing the new event further to another service like lammbda in our case. The view type should be **"New and Old Image"** which will make sure that the lambda function is trigger with new and old images.

5.6 GenAI Models

The genAI models are developed by the vendors and are added in AWS bedrock which provides a single API to run the prompt. The models are not generally available with the access to the IAM yet they need to explicitly enabley Bedrock console. Titan Text G1 - Premier and Mistral Large (24.02). The models are configured with Randomness, diversity and length with below configuration and I have kept it default to the bedrock model configuration only.

- Temperature - 0.7
- Top P - 0.9 (top p chooses the smallest set of word which is used in nucleus sampling.
- Response length - 512 (Maximum no of character)

5.7 IAM and Logging

IAM permissions for Lambda to access Dynamodb table is an important consideration in implementing the solution. Lambda must be given get_item list_item and put_item permission on the lambda role to manage the operations.

The responses are recorded in Cloudwatch form both the models. Multiple lambda functions with alias and version were used with a single log group to centralize the log collection of the responses from the genAI model.

6 Evaluation

The test evaluation is divided into three major parts, focusing on critical metrics and qualitative aspects to compare the generative AI model responses. The evaluation framework is designed to provide a comprehensive assessment of the models' performance under different scenarios:

6.1 Experimental Setup

The evaluation involved conducting 100 iterations to generate responses, with 48 iterations recorded for detailed analysis. The breakdown of iterations is as follows:

Model Distribution:

1. 24 Iterations for Mistral AI

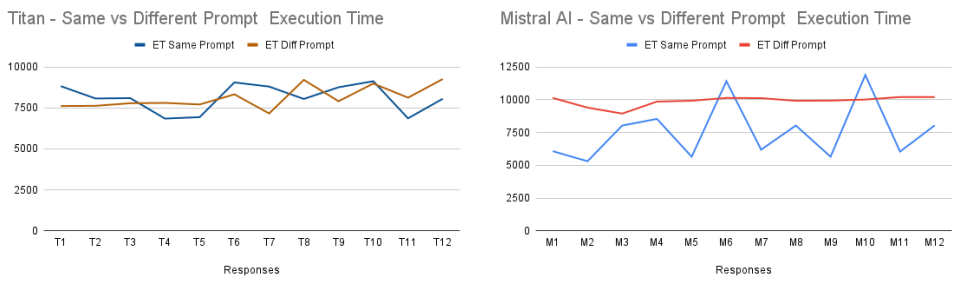

Figure 6: Titan Indentical Vs Different Prompt Execution Time

2. 24 Iterations for Amazon Titan

Prompt Variations:

3. 12 Identical Prompts: Half of the iterations used the Identical prompt with consistent flags and transaction details. This setup ensures a controlled environment to evaluate response consistency and quality for identical scenarios.

4. 12 Unique Prompts: The remaining iterations used entirely distinct prompts, varying in transaction location, amount, and fraud flags. This tests the models' adaptability and ability to handle diverse scenarios.

6.2 Execution Time

This aspect measures the efficiency of each generative AI model by recording the time taken to receive a response after sending a prompt. Execution time is a crucial metric for determining the practicality of using these models in real-time applications. The test focuses on:

Average Latency: Time taken from prompt submission to receiving the response. Variability Across Iterations: Consistency in response times, particularly for high-complexity prompts. Model Comparison: Direct comparison of execution times between Amazon Titan and Mistral AI under identical conditions.

6.3 Cost Analysis

This aspect evaluates the cost-effectiveness of the models, considering the pricing structure based on input and output tokens. The evaluation excludes the knowledge base, as it utilizes a vector database that charges uniformly for all requests. Specific considerations include:

- **Token Costs**: Analysis of input (prompt) and output (response) tokens to calculate the cost per request.
- **Cost Efficiency**: Comparison of the cost incurred for similar tasks on both models.
- **Cost-Performance Tradeoff**: Balancing execution costs against response quality and comprehensiveness.

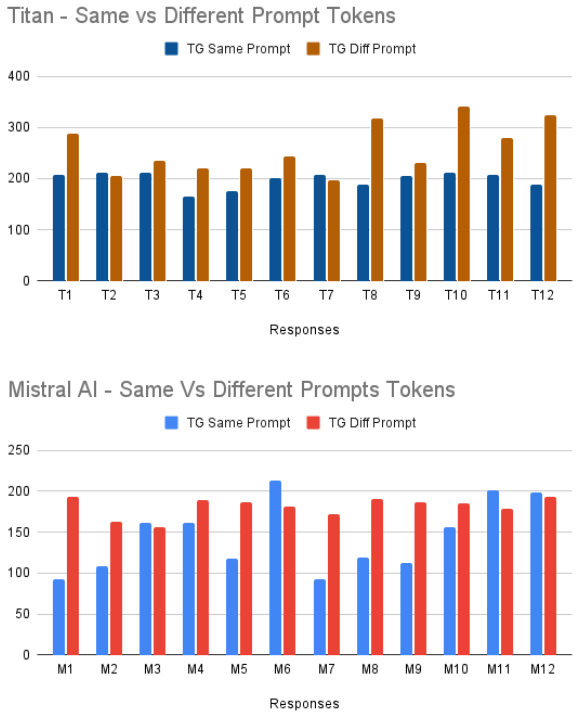

Figure 7: Titan Vs Mistral - Overlapped bar graph demonstrating the token generated on Titan and Mistral with Identical and Different prompts for 12 iterations each case

6.4 Response Differentiation

This aspect focuses on the models' ability to differentiate and provide comprehensive analyses of fraudulent transactions. Key criteria for evaluation include:

Number of Flags Identified: The extent to which each model identifies fraud indicators, such as mismatched billing and shipping addresses, high-risk categories, or suspicious IP addresses. **Preventive Measures**: The quality and relevance of the suggested strategies for mitigating similar fraud in the future. **Cosine Similarity**: Semantic similarity between responses when addressing similar prompts, providing insight into consistency. Suggestions for Future: The clarity and actionable nature of forward-looking recommendations. **Readability Score**: Assessing the ease of understanding the response, using standard readability metrics.

6.5 Token Generation

Tokens are an important part of evaluation as these models are capable to consume 32k tokens in a single prompt but the output can vary depending upon the prompt and the context from the knowledge base. The more tokens it generates with more information from the knowledge base, can be used further to mitigate and anlyse the fraud transcations better

6.6 Experiment 1 / Similar prompt to GenAI

- Titan Model: The similar prompts when ran through titan model showed very stable results keeping the average to 8121.86 ms for all the 12 prompts ran.

Aspect	Titan Response	Mistral Response
Number of Flags	Lists 6 flags with detailed explanations.	Lists 6 flags, but provides less detail for each.
Preventive Measures	Offers 6 detailed measures with actionable steps (e.g., monitoring systems, address verification).	Provides 5 preventive measures, summarized (e.g., mentions "xFraud" tools without detailed specifics).
Cosine Similarity of RAG	0.85 (shared context)	0.85 (shared context)
Suggestions for Future	Specific and actionable recommendations tailored to the transaction scenario.	General suggestions, without tailoring to the specific transaction context.
Readability Score	˜50 (Moderate)	˜70 (Easier)

Table 1: Comparison of Titan and Mistral Responses based on Flags, cosine similarity. The responses are generated from multiple public LLM models like Chatgpt, Gemini and Azure Copilot

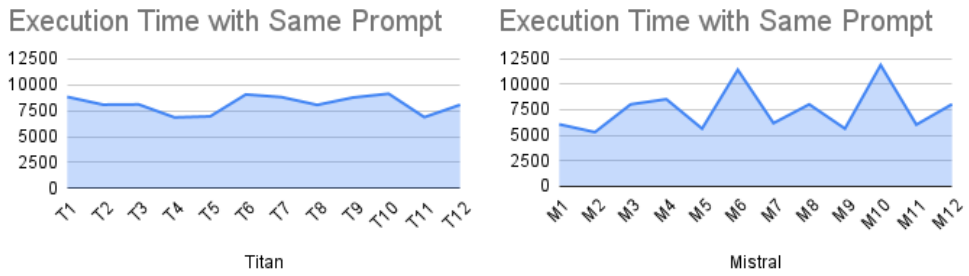

Figure 8: **Experiment 1**: Comparison of execution times for the Identical prompt across Titan and Mistral models. The results indicate greater stability in Titan's performance, while Mistral exhibits notable variability in execution times.

- Mistral Model: For the mistral AI an average of 7570.68 ms was seen which makes mistral model more efficient but but the amount of variations in the response were not stable

6.7 Experiment 2 / Different prompt to GenAI

The second iteration as seen in Fig 5 and Fig 8 of tests were performed 12 times for each models with different prompts provided with variable flags, amount, location and customer information. The results indicate dynamic responses from Titan and very stable responses from Mistral. This shows clearly that Titan model learns overtime and get new information from knowledge base while Mistral runs the similar algorithm of NLP and BERT to gather responses.

6.8 Experiment 3 / Analyzing Responses using LLM

The evaluation metrics/aspects for the response generated from both the models were compared on multiple LLM models which is ChatGPT 4, Gemini and Azure Copilot.

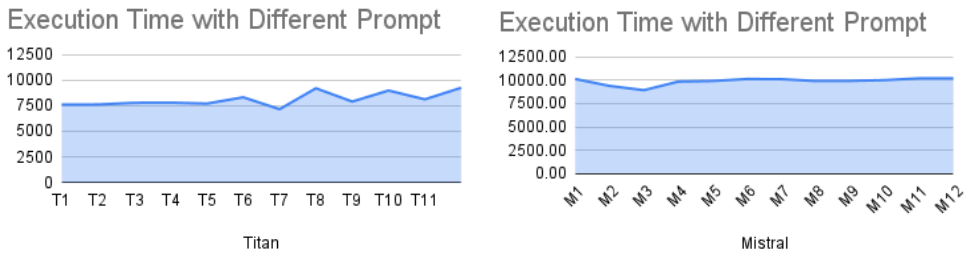

Figure 9: **Experiment 2**: Comparison of execution times for the different prompt across Titan and Mistral models. The results indicate variable results in Titan which is good but stability in Mistral's performance. Thus, mistral gave similar responses for most of the prompts.

Each genai model were given 20 different prompt's responses to find out the number of flags which came as an average of 6 flags for both the models.

1. For Preventive measures, titan responded with more actionable steps as compared with Mistral AI. Measures like monitoring systems and address verification were clearly visible while in Mistral the context from RAG was not scene

2. Cosine similarity which is an overlap of words came nearly similar for both the models.

3. Readability score for titan was moderate with 50 and for mistral it was 70 with an ease of reading and understanding the analyses.

6.9 Discussion

Upon looking into the results of execution time, Titan models are more quicker to provide the responses with context to the retrievals from vector database as in comparison with the Mistral AI. For Identical and different prompts provided to Titan, the results varied over prompts, this provides insights that the model is continuously improving with the vector database and trying to give more details.

On the other hand, mistral AI execution time for different prompts coming very similar but the Identical prompts are its varying heavily thus uncertainty in the model is seen.

The findings from the experiments highlight several important insights into the performance and usability of the Amazon Titan and Mistral AI models in analyzing fraudulent transactions using Retrieval-Augmented Generation (RAG). While the experiments provide a strong foundation for comparing these models, there are areas of improvement and limitations in the design that warrant further discussion.

Strengths of the Study Comprehensive Metrics Evaluation: The experiments evaluated the models on critical metrics such as execution time, token generation, comprehensiveness of responses, and cost. This multidimensional analysis provided a well-rounded understanding of their performance.

Integration with RAG: By leveraging RAG, the study demonstrated how integrating a knowledge base can enhance the contextual accuracy of model outputs, particularly for domain-specific tasks like fraud analysis and text summarization. Titan's superior ability to retrieve and incorporate relevant information validated the efficacy of this approach.

Cost Analysis: The cost evaluation highlighted the practical implications of deploying these models at scale. The findings clearly showed that Titan is more cost-efficient, especially for input and output token generation, making it a preferred choice for user and systems applications.

7 Conclusion and Future Work

Generative AI (GenAI) models have proven effective, yet they often struggle with domain-specific knowledge. The research is implmenting Retrieval Augmented Generation (RAG) has significantly enhanced these models by enabling them to retrieve relevant context from a knowledge base, improving their accuracy and utility in specialized applications.

In this study, we compared Amazon Titan and Mistral AI across several critical factors, including execution time, token efficiency, cost, and the comprehensiveness of their responses. Amazon Titan consistently outperformed Mistral AI, delivering more accurate and contextually relevant responses at a lower cost. Its ability to provide detailed and actionable insights makes it particularly suited for fraud mitigation and other high-stakes applications.

The scope of this analysis can be expanded to include additional domains and industries. While these models are pre-trained, future work could explore fine-tuning them further using AWS Bedrock's fine-tuning tools and larger datasets. While this would increase research costs, the potential improvements in performance and accuracy would make it a worthwhile endeavor.

Moreover, as AWS Bedrock continues to recognize and integrate new models monthly, future comparisons could include these emerging options. Introducing advanced evaluation metrics, particularly focused on the integration and performance with knowledge bases, would provide a more robust framework for assessing these models.

References

Arora, D., Kini, A., Chowdhury, S. R., Natarajan, N., Sinha, G. and Sharma, A. (2023). Gar-meets-rag paradigm for zero-shot information retrieval, *arXiv preprint arXiv:2310.20158* .

Bian, N., Liu, P., Han, X., Lin, H., Lu, Y., He, B. and Sun, L. (2024). A drop of ink may make a million think: The spread of false information in large language models, *Institute of Software, Chinese Academy of Sciences* .

Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., van den Driessche, G., Lespiau, J.-B., Damoc, B., Clark, A. et al. (2022). Improving language models by retrieving from trillions of tokens, *arXiv preprint arXiv:2112.04426* .

Generative Artificial Intelligence: Prospects for Banking Industry (n.d.). *International Journal of Research in Engineering, Science and Management* . Accessed: 2024-12-09. **URL:** *https://journal.ijresm.com/index.php/ijresm/article/view/2969/2995*

Hacker, P., Engel, A. and Mauer, M. (2023). Regulating chatgpt and other large generative ai models, *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency (FAccT '23)*, ACM, New York, NY, USA, pp. 1112–1123.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L. and Chen, W. (2021). Lora: Low-rank adaptation of large language models, *arXiv preprint arXiv:2106.09685* .

Jiang, Z., Xu, F., Gao, L., Sun, Z., Liu, Q., Dwivedi-Yu, J., Yang, Y., Callan, J. and Neubig, G. (2023). Active retrieval augmented generation, *arXiv preprint arXiv:2305.06983* .

Kim, G., Kim, S., Jeon, B., Park, J. and Kang, J. (2023). Tree of clarifications: Answering ambiguous questions with retrieval-augmented large language models, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, pp. 996–1009.
URL: *https://aclanthology.org/2023.emnlp-main.63*

Lester, B., Al-Rfou, R. and Constant, N. (2021). The power of scale for parameter-efficient prompt tuning, *arXiv preprint arXiv:2104.08691* .

Li, H., Su, Y., Cai, D., Wang, Y. and Liu, L. (2022). A survey on retrieval-augmented text generation, *arXiv preprint arXiv:2202.01110* .
URL: *https://arxiv.org/abs/2202.01110*

Liu, X., Ji, K., Fu, Y., Tam, W. L., Du, Z., Yang, Z. and Tang, J. (2021). P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks, *arXiv preprint arXiv:2110.07602* .

NVIDIA (2024). What is retrieval-augmented generation? Accessed: 2024-12-09.
URL: *https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/*

Prabhune, S. and Berndt, D. J. (2024). Deploying large language models with retrieval augmented generation, *arXiv* .
URL: *https://arxiv.org/pdf/2411.11895*

Sallam, M., Khalil, R. and Sallam, M. (2024). Benchmarking generative ai: A call for establishing a comprehensive framework and a generative aiq test, *Mesopotamian Journal of Artificial Intelligence in Healthcare* pp. 69–75.

Singh, B. (2024). Generative artificial intelligence: Prospects for banking industry, *Chief Manager Research, State Bank of India* .

Trivedi, H., Balasubramanian, N., Khot, T. and Sabharwal, A. (2023). Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions, *arXiv preprint arXiv:2212.10509* .

Yu, H., Gan, A., Zhang, K., Tong, S., Liu, Q. and Liu, Z. (2024). Evaluation of retrieval-augmented generation: A survey, *arXiv preprint arXiv:2405.07437* .

Zhao, P., Zhang, H., Yu, Q., Wang, Z., Geng, Y., Fu, F., Yang, L., Zhang, W. and Cui, B. (2024). Retrieval-augmented generation for ai-generated content: A survey, *arXiv preprint arXiv:2402.19473* .