# SMART CHATBOTS & API ROUTING VIA LLMS

PROJECT REPORT

SUBMITTED TO

SVKM'S NMIMS (Deemed to be) UNIVERSITY

IN PARTIAL FULFILLMENT FOR THE DEGREE OF

**MASTER OF SCIENCE**
**IN**
**STATISTICS & DATA SCIENCE**

BY

**Khushi Jain**

NMIMS NILKAMAL SCHOOL OF MATHEMATICS,
APPLIED STATISTICS & ANALYTICS
V. L. Mehta Road, Vile- Parle (West)
Mumbai – 400056

APRIL, 2025

# SMART CHATBOTS & API ROUTING VIA LLMS

PROJECT REPORT

SUBMITTED TO

SVKM'S NMIMS (Deemed to be) UNIVERSITY

IN PARTIAL FULFILLMENT FOR THE DEGREE OF

**MASTER OF SCIENCE**
**IN**
**STATISTICS & DATA SCIENCE**

BY

**Khushi Jain**

NMIMS NILKAMAL SCHOOL OF MATHEMATICS,
APPLIED STATISTICS & ANALYTICS
V. L. Mehta Road, Vile- Parle (West)
Mumbai – 400056

APRIL, 2025

# SMART CHATBOTS & API ROUTING VIA LLMS

PROJECT REPORT

SUBMITTED TO

SVKM'S NMIMS (Deemed to be) UNIVERSITY

IN PARTIAL FULFILLMENT FOR THE DEGREE OF

**MASTER OF SCIENCE**
**IN**
**STATISTICS & DATA SCIENCE**

**BY**

**Khushi Jain**

**Program Director (Statistics)**
**(Prof. Sunil S. Shirvaiker)**

**Dean NSOMASA**
**(Dr. Sushil Kulkarni)**

NMIMS NILKAMAL SCHOOL OF MATHEMATICS, APPLIED STATISTICS & ANALYTICS

NMIMS NILKAMAL SCHOOL OF MATHEMATICS,
APPLIED STATISTICS & ANALYTICS
V. L. Mehta Road, Vile- Parle (West)
Mumbai – 400

# CERTIFICATE

This is to certify that work described in this thesis entitled **SMART CHATBOTS & API ROUTING VIA LLMS** has been carried out by Khushi Jain under my supervision. I certify that this is her bonafide work. The work described is original and has not been submitted for any degree to this or any other University.

**Date:**

**Place:**

08-04-2025

X [illegible]

Vikas Rathod
CTO
Signed by: 60e7f9fa-b3a7-44be-8b88-b2ee9d420da6

| | |
|---|---|
| **Internal Mentor** | **Company Mentor** |
| Prof. Sunil S. Shirvaiker | Mr. Vikas Rathod |
| **Date:** | **Date:** |

NIfe Healthcare Pvt Ltd - Ground Floor, 36, Infantry Rd, Tasker Town, Shivaji Nagar, Bengaluru, Karnataka 560

# ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to **Mr. Vikas Rathod** for his continuous support throughout the project as well as patience, motivation, and immense knowledge. Their guidance helped me throughout my project and for the writing of this report. I could not have imagined having a better advisor and mentor for this project.

My sincere thanks also goes to **Prof. Sunil S. Shirvaiker**, who helped me throughout the course of my project. His tremendous experience and crucial remarks were really helpful in shaping our final report and presentation.

I would also like to give a special thanks to **Mr. Yagyam Batra** who has immensely helped me putting down this work in a correct manner and supported throughout.

Last but not least, I would like to express my deepest gratitude to **my family and other friends**. This project would not have been possible without their warm love, continued patience, and endless support.

I owe my wholehearted thanks and appreciation to the entire staff of the company for their cooperation and assistance during the course of my project

# CONTENTS

# 1. ABSTRACT

The exponential growth in the fields of Artificial Intelligence (AI) and Natural Language Processing (NLP) has opened the door for creating systems that can replicate human-level understanding, reasoning, and decision-making. Central to this evolution are Large Language Models (LLMs) like Gemini, which have revolutionized how machines interpret and generate human language. This report presents a detailed exploration of two AI-powered projects developed individually, each targeting a specific real-world problem and addressing it through modern NLP pipelines and intelligent agent design.

The first project, titled **NifeCare Chatbot**, was designed for users of the Nife Healthcare application. Nife Healthcare provides a comprehensive health and wellness prepaid card integrated with a large support ecosystem including insurance partners, hospitals, OPD centers, and pharmacies. The application may pose usability challenges for first-time or non-technical users. To address this, the chatbot was built to understand and respond to user queries derived from the Nife user manual. The project utilized LangChain for pipeline management, Sentence Transformers for embedding generation, ChromaDB for vector storage, and Gemini 1.5 Pro for generating intelligent, context-aware responses. Using the Retrieval-Augmented Generation (RAG) framework, the chatbot identifies the top relevant document chunks (k=8) using KNN search, and generates answers that are semantically aligned with the user's query.

The second project, titled **Dynamic API Agent using Gemini and Tool Mapping**, explores the concept of **agentic LLM behavior**. The system functions as a question router that dynamically determines the nature of the user's query—be it a mathematical problem, date-related question, or functional task—and forwards it to the appropriate API based on Gemini AI's interpretation. Instead of hardcoding rules, the AI agent intelligently selects the relevant function from a tools list and executes it using a function map. For example, a query like "What is 5+9?" will be automatically routed to a calculator API, executed, and returned to the user—without requiring any manual configuration. This highlights how LLMs can serve as cognitive middleware that interfaces between human language and executable logic, offering tremendous automation capabilities in enterprise systems.

Both projects embody modern practices in AI-driven development and are built using Google Colab and VS Code environments. The chatbot project was completed within a span of 2.5 months, while the API Agent project is ongoing with a functioning prototype. This report covers the motivation behind each project, design choices, technological stack, theoretical background, implementation workflows, and the respective outcomes. It also discusses limitations, potential enhancements, and broader implications in domains such as healthcare support systems and intelligent agent-based automation.

By integrating powerful LLMs with real-time data pipelines and intelligent routing, these projects demonstrate the growing potential of AI in simplifying complex user experiences, enhancing automation, and creating scalable digital ecosystems. Their significance lies not just in functionality but in paving the way for adaptive, responsive systems that can reason, retrieve, and act intelligently in diverse user-driven scenarios.

# 2. INTRODUCTION

The convergence of deep learning and natural language processing has led to groundbreaking advancements in how machines understand and interact with human language. Central to this revolution are **Large Language Models (LLMs)**, which now serve as the core building blocks of intelligent systems like chatbots, autonomous agents, and dynamic software assistants. With the rise of robust orchestration frameworks like **LangChain**, the potential of LLMs is further amplified—allowing them not just to respond to queries, but to think, decide, and act within multi-step workflows using external tools and APIs. This paradigm shift from static chatbot frameworks to agentic systems capable of intelligent reasoning is the foundation for both **Project 1 (NifeCare Chatbot)** and **Project 2 (Dynamic API Agent)**.

The goal of both projects was to build intelligent, modular, and scalable LLM-powered systems that could either:

- Accurately answer domain-specific user queries using internal documentation (Project 1), or
- Dynamically select and execute relevant APIs/functions based on natural language prompts (Project 2).

To properly understand how these systems function, it is essential to dive deep into the theoretical underpinnings of the tools and techniques used. The following subsections explain the key concepts that power these innovations.

## 2.1 Large Language Models (LLMs)

**LLMs** are advanced neural networks, often with billions of parameters, trained on diverse textual corpora to learn the structure, semantics, and conceptuality of language. They are typically based on the **Transformer architecture**, introduced by Vaswani et al. (2017), which utilizes self-attention mechanisms to process tokens in parallel while maintaining their contextual relationships.

### Key Theoretical Concepts:

- **Tokenization**: Breaking down input text into units that the model can process (e.g., words, sub words, or characters).
- **Self-Attention**: Allows the model to weigh the importance of each word relative to others in a sentence or paragraph.
- **Positional Encoding**: Since Transformers are not inherently sequential, positional encodings are added to preserve word order.
- **Pre-training and Fine-tuning**: Models are pretrained on massive corpora and then fine-tuned on specific tasks for greater accuracy.
- **Next-token Prediction**: The core function during training—predicting the next word in a sentence based on the context.

### Applications:

- Text completion
- Summarization

- Translation
- Reasoning
- Dialogue generation
- Code interpretation

LLMs like **Gemini 1.5 Pro**, **GPT-4**, and **Claude** are capable of zero-shot and few-shot learning, meaning they can handle novel tasks without specific retraining. These models power both of our projects by enabling them to comprehend, retrieve, and respond with domain-specific intelligence.

## 2.2 LangChain Framework

**LangChain** is a compostable framework designed to build applications with LLMs by chaining together components like memory, prompts, tools, agents, retrievers, and more. It simplifies the development of advanced AI systems by offering:

- **Modularity**: Each component (retriever, memory, chain, etc.) is independent and can be configured individually.
- **Integrations**: Connects with LLMs like OpenAI, Gemini, Cohere, and Anthropic, and tools like ChromaDB and Pinecone.
- **Tool Use**: Allows LLMs to use functions, APIs, calculators, search engines, etc., autonomously.

### Core Concepts in LangChain:

- **Prompt Templates**: Dynamic, reusable input formats.
- **Chains**: Logical sequences of operations such as: Prompt → LLM → Output Parsing.
- **Memory**: Enables chatbots to retain context across sessions.
- **Agents**: LLM-powered systems that choose tools or actions dynamically.
- **Toolkits**: Collections of callable tools/functions used by agents.

LangChain bridges the gap between raw LLMs and usable applications, making it instrumental in both projects.

## 2.3 Retrieval-Augmented Generation (RAG)

**RAG** is a hybrid technique combining:

- **Retrieval**: Extracting relevant documents from a knowledge base using embedding's and similarity search.
- **Generation**: Using an LLM to synthesize a response from the retrieved documents.

This overcomes LLMs' limitation of not having real-time or domain-specific memory, allowing them to answer factually accurate, context-specific questions.

### Key Steps in RAG:

- Document chunking and embedding using Sentence Transformers (e.g., SBERT).
- Storing vector embedding's in a vector database (e.g., ChromaDB).

- Querying top-k similar chunks at runtime.
- Feeding retrieved chunks into LLM prompt for grounded answer generation.

In **Project 1**, this technique enabled the chatbot to provide real-time answers based on internal company documentation (NifeCare manual), making it highly contextual and accurate.

## 2.4 Agentic APIs and Autonomous Decision-Making

Traditional software relies on rigid logic to invoke APIs. However, **Agentic APIs** enable a system to autonomously decide:

- **Which tool to use**
- **What function to call**
- **What input to provide**

An **LLM Agent**, using reasoning capabilities (e.g., ReAct or Toolformer patterns), evaluates the user's intent and dynamically chooses the best-suited API from a toolset. This marks the evolution from “chatbots” to intelligent **task-solving agents**.

### Benefits:

- Eliminates need for hardcoded workflows
- Allows scalability across multiple functions/APIs
- Enables adaptive behavior based on diverse user inputs

In **Project 2**, an LLM-based agent selects from a mapped list of tools (calculator, date/time, BMI calculator, etc.) based on the query and executes it using an appropriate interface. This adds modularity and intelligence to otherwise static interfaces.

## 2.5 Chatbots and Conversational Agents

Chatbots today have moved far beyond button-based interfaces. With **Conversational AI**, they now support:

- Multi-turn dialogue
- Context retention
- Sentiment understanding
- Information retrieval
- Dynamic routing to agents/APIs

Project 1 builds such a chatbot using LangChain, Gemini, ChromaDB, and RAG to assist users of **NifeCard**—a health and wellness product platform—with real-time answers derived from internal documents.

## 2.6 Gemini 1.5 Pro

**Gemini 1.5 Pro**, developed by Google DeepMind, is one of the most advanced LLMs available in 2024–2025. It supports:

- Context windows up to **1 million tokens**
- Seamless tool integration
- Zero-shot reasoning
- Visual and document understanding
- High accuracy with low latency

Used in both projects, Gemini provides robust comprehension and tool-use reasoning. It is responsible for both:

- Answer generation in Project 1 (based on RAG context),
- Tool selection and execution in Project 2 (via LangChain agent).

## 2.7 Additional Theoretical Pillars

Below are additional concepts that provide foundational theory for the projects:

- **Sentence Embedding's**: Semantic vector representations used for document retrieval (SBERT, MiniLM, etc.).
- **Vector Similarity Search**: Algorithms like Cosine Similarity and FAISS for finding similar document chunks.
- **Toolformer**: The idea that LLMs can self-learn how to use APIs/tools given few examples.
- **ReAct (Reason + Act)**: A prompting strategy where the model is asked to first reason, then act (used for tool use and planning).
- **Function Calling**: API interface design where model output is structured as a JSON to call functions.

# 3. RATIONAL

The increasing reliance on intelligent systems across industries necessitates a paradigm shift from passive information processing to **contextual understanding and actionable output**. The rationale behind undertaking the two interconnected AI-driven projects—**Project 1: NifeCare Chatbot** and **Project 2: Dynamic API Agent using Gemini and Tool Mapping**—is to demonstrate the transformative potential of combining **retrieval-augmented generation (RAG)** with **agentic language models**. This approach not only makes large language models (LLMs) more useful but also deeply embedded in real-world use cases, where context and action matter more than raw generative capability.

## 3.1 Real-World Gap between AI Promise and Application

Although modern LLMs have demonstrated remarkable capabilities in generating human-like responses, the transition from **novelty to utility** is where many systems fail. The gap lies in two fundamental shortcomings:

1. **Lack of Grounded Knowledge** – Most LLMs are trained on internet-scale corpora, but lack access to **organization-specific, real-time** information. This limits their trustworthiness in enterprise environments.
2. **Limited Interactivity** – Traditional LLM-based chatbots can only generate responses but cannot perform dynamic computations, access tools, or modify actions based on user intent.

These two projects aim to address these gaps by:

- Using **document-based RAG pipelines** (Project 1) to create domain-specific chatbots that are factually grounded.
- Leveraging **function calling and agentic reasoning** (Project 2) to empower LLMs to dynamically decide what action to take, map it to an API, execute it, and return results.

## 3.2 Enterprise Motivation (Project 1: NifeCare Chatbot)

In healthcare, particularly within platforms like **NifeCard**, users often encounter difficulty understanding the extent and method of benefits such as:

- Accessing the ₹3,00,000 group health insurance
- Finding eligible cashless hospitals
- Understanding gym or wellness partnerships
- Navigating insurance claim processes

These platforms offer a multitude of services, which may overwhelm even tech-savvy users, especially in emergency situations. Static FAQs or even scripted chatbots fail to cover **long-tail, user-specific queries**, leading to poor user satisfaction and low feature adoption.

By using a **LLM + RAG-based chatbot**, the system can:

- Parse through lengthy PDF manuals, guidelines, and documentation
- Extract and deliver relevant answers to user-specific questions
- Handle paraphrased, ambiguous, or incomplete queries through contextual understanding
- Be updated in real-time by modifying documents, not retraining models

Moreover, such a chatbot becomes an integral part of **digital transformation**—automating user assistance, reducing operational costs, and offering a highly accessible self-service option to cardholders.

## 3.3 Agentic Thinking (Project 2: Dynamic API Agent)

Project 2 takes this one step forward: instead of merely responding to questions, the LLM is empowered to **solve a problem by taking action**. This aligns with the broader concept of **LLM agents**, where the system is aware of the tools at its disposal and can:

- Break down user input
- Choose the correct function or API
- Supply parameters or arguments
- Interpret and return a usable output

For example:

- **Input:** "Convert 45°F to Celsius"
- **Agent Reasoning:** Identify it as a temperature conversion task
- **Action:** Call convert_temperature(fahrenheit=45)
- **Output:** 7.2°C

These agents mimic the cognitive workflow of a human assistant and apply **tool reasoning frameworks** like:

- **ReAct**: Reasoning + Acting
- **Toolformer**: Self-learning tool use
- **Function Calling (OpenAI/Gemini)**: External utility integration

Thus, this project does not merely build a chatbot; it **orchestrates logic, tools, and language** under a unified intelligent framework. It gives rise to a general-purpose assistant capable of interacting with computational backends using simple natural language queries.

## 3.4 Theoretical Justification

Both projects are supported by cutting-edge research in:

- **RAG** (Lewis et al., 2020): Retrieval-augmented generation minimizes hallucination in LLMs by grounding output in retrieved documents.
- **Toolformer** (Schick et al., 2023): Demonstrates that LLMs can learn when and how to use tools from examples.
- **LangChain Agents**: Extend LLM capabilities through action-oriented chains, making interaction programmable, modular, and goal-driven.
- **Sentence Transformers** (Reimers & Gurevych, 2019): Enable semantic search and document embedding for accurate retrieval.

These techniques form the spine of the projects, making them academically sound and industry-ready.

## 3.5 Generalizability across Domains

The solutions aren't limited to NifeCard or basic calculators. The underlying architecture is **modular and domain-agnostic**, making it deployable across:

- **Banking**: Querying interest rates, loan conditions, performing EMI calculations
- **E-commerce**: Order tracking, discount eligibility, price comparison
- **Education**: Exam scheduling, grading, academic resource retrieval
- **Legal/HR**: Policy document comprehension, leave balance calculations

This flexibility amplifies the real-world utility and ROI of these systems, especially for organizations seeking automation without rebuilding tools from scratch.

## 3.6 Innovation, Simplicity, and Practicality

While AI research often focuses on pushing boundaries, this project emphasizes **balance**—choosing methods that are:

- **Effective yet simple**: No need for expensive GPU clusters or multi-agent learning.
- **Lightweight and adaptable**: Built on Colab using accessible tools like ChromaDB, LangChain, and Gemini API.
- **Cost-efficient**: Easily deployable by startups and medium-sized enterprises.

The goal is not to build an over-engineered product but a **working prototype** that demonstrates:

- Clarity of purpose
- Smart orchestration of existing tools
- A direct path from user intent to task execution

## 3.7 Future Vision and Agent Autonomy

The true vision lies in pushing toward **autonomous task execution**. As models improve:

- The same framework can evolve into **multi-step task solvers**
- Agents could learn task hierarchies, self-monitor outputs, or even request clarification
- Tool libraries can be expanded to offer financial, healthcare, or logistics capabilities

Ultimately, these systems aim to deliver a **natural language interface for computing itself**—where a user simply says what they need, and the AI figures out how to get it done.

## 3.8 Scope

The scope of this report encompasses the design, development, implementation, and potential impact of two independent but thematically aligned projects — **Project 1: NifeCare Chatbot** and **Project 2: Dynamic API Agent**. These projects are centered on leveraging the capabilities of **Large Language Models (LLMs)** in the field of **AI-driven information retrieval, decision-making, and tool invocation**.

Both projects are designed to demonstrate practical applications of the latest innovations in **Natural Language Processing (NLP)**, including **Retrieval-Augmented Generation (RAG)**, **Agentic AI**, and **LLM-based orchestration of APIs**, while maintaining modularity and future scalability.

### Project 1: NifeCare Chatbot – Scope

- **Domain**: Healthcare and Wellness Card Support System
- **Dataset**: A detailed PDF manual provided by NifeCard (proprietary user guide of the Health & Wellness RuPay Prepaid Card)
- **User Group**: Cardholders, customer support teams, and potential customers seeking information
- **Model**: Gemini 1.5 Pro (for language generation), SBERT (for semantic embeddings)
- **Frameworks**: LangChain, ChromaDB, Python, Google Colab
- **Functional Scope**:
    - Ingestion of large documents in various formats (PDF, DOCX, TXT)
    - Text chunking and semantic embedding of content
    - Vector storage in ChromaDB for efficient retrieval
    - Top-k similarity search (K=8) to fetch the most relevant chunks
    - Grounded answer generation using Gemini 1.5 Pro via LangChain
    - Chat interface for real-time user interaction
- **Exclusions**:
    - Does not yet support voice-based interaction
    - Limited to the scope of uploaded documents (i.e., no external web search)
    - Only handles queries related to uploaded documents

## Project 2: Dynamic API Agent – Scope

- **Domain**: AI-Augmented Task Execution & Intelligent Tool Mapping
- **Environment**: Google Colab (Python), Gemini 1.5 Pro
- **Target Users**: Developers, AI integrators, automation engineers, and users seeking dynamic assistance via chatbots
- **APIs Handled**: Basic mathematical calculations, statistical formula placeholders, time/date functions (currently); can be extended to complex domains
- **Agent Mechanism**:
    - Uses Gemini 1.5 Pro to parse intent
    - Maps query to appropriate function/tool based on metadata
    - Invokes tool and returns result in real-time
- **Functionality Covered**:
    - Semantic intent extraction from free-form user queries
    - Dynamic tool invocation using LangChain's Tool schema
    - Tool selection strategy using zero-shot prompting and metadata parsing
    - Output formatting and response delivery via conversational flow
- **Limitations**:
    - Currently supports only basic tool usage (i.e., calculator-like functions)
    - Does not yet handle multi-step planning or sequential tool usage
    - No user authentication or logging modules implemented

## Common Scope Highlights

- **LLM Utilization**: Both projects showcase the power of Gemini 1.5 Pro in performing reasoning, generation, and intelligent mapping based on user prompts.
- **Modular Design**: Each component is designed to be pluggable and upgradable—embedding models, vector stores, LLM APIs, and even UI modules can be replaced or extended.
- **Research-Driven Development**: The projects are heavily influenced by academic literature and engineering practices around RAG pipelines, LangChain, ReAct frameworks, and Toolformer.
- **Evaluation Plan**:
    - Currently undergoing functional testing and benchmarking
    - Qualitative assessment of accuracy, relevance, and user satisfaction planned in future phases
- **Scalability**:
    - Designed with expansion in mind—can scale to new domains like finance, education, legal, and more with minimal changes
    - Can integrate with external APIs for richer interactions (weather, flight booking, sentiment analysis, etc.)

Beyond the engineering objectives, the projects were also rooted in a deeper academic and pedagogical philosophy—one that recognizes the importance of accessibility, transparency, and real-world relevance in the rapidly evolving world of artificial intelligence. In today's digital ecosystem, the deployment of AI-driven solutions is no longer restricted to large tech companies or elite research labs. Open-source tools, pre-trained models, and flexible platforms have democratized the development of AI applications. However, the actual understanding of how these systems operate under the hood—how a chatbot retrieves answers from documents, how a query is routed to a relevant API via an agent, how embeddings function in vector spaces, and how language models interact with structured tools—is still limited in most educational and business contexts. These projects were thus conceived not only to solve practical use-cases but also to expose the layered mechanics of LLMs, LangChain pipelines, RAG architectures, and agentic reasoning to a wider audience, particularly to students, budding developers, and professionals with non-AI backgrounds.

By constructing the NifeCare Chatbot and the Dynamic API Agent from scratch, this project series becomes a live tutorial in the design of intelligent applications. It highlights the fact that modern LLM-powered tools are not monolithic but modular systems composed of interoperable parts—document loaders, text chunkers, embedding generators, vector databases, and inference engines. Each of these components plays a distinct role in the pipeline and can be optimized, swapped, or extended depending on the domain-specific need. For example, in the chatbot project, the decision to use SentenceTransformers for generating embeddings and ChromaDB for vector storage wasn't arbitrary. It was an intentional choice based on factors like speed, compatibility with LangChain, scalability, and ease of integration. Similarly, in the API Agent project, the use of a function map and a Gemini agent to dynamically dispatch API calls based on user intent demonstrates the concept of "agentic orchestration," a paradigm that is now central to next-gen AI systems.

Additionally, these projects address a growing concern in the AI industry—the ability of models to remain interpretable, context-aware, and utility-driven. With tools like Gemini 1.5 Pro integrated within LangChain agents, the project illustrates how LLMs can move beyond static chat completion and into actionable workflows where each query is contextually analyzed, mapped to a tool, and executed with measurable outputs. This is particularly significant in the second project where the agent chooses the right API (e.g., calculator API for math queries) based on task analysis. This practical demonstration of reasoning + acting, sometimes referred to in literature as the ReAct framework, showcases how cognitive chains of thought can be translated into machine actions—thus bridging the gap between language and execution.

There is also a significant emphasis in both projects on future extensibility and real-world deployability. Unlike traditional academic experiments that often remain theoretical, these systems were built with real use-cases in mind: the NifeCare Chatbot aims to assist consumers of a digital healthcare service app, while the API Agent serves as a meta-layer to simplify access to backend tools and microservices. These are scalable ideas that can be generalized and reused across domains—from edtech to fintech, from customer service to enterprise automation. Moreover, since the technologies used (LangChain, Gemini, ChromaDB, SentenceTransformers) are either open-source or have extensive documentation and community support, the barrier to entry is minimal, making these solutions ideal for rapid prototyping and educational purposes.

Finally, these projects serve a reflective role in the AI design process—showing that AI is not magic, but structured problem-solving. Every step taken, from designing workflows to choosing hyperparameters to structuring prompts and tool responses, was a learning experience grounded in core concepts like retrieval-augmented generation (RAG), few-shot prompting, embedding mathematics, and modular software engineering. As such, the value of these projects lies not just in what they do, but in what they teach. For students and developers aiming to break into the AI field, this layered understanding is invaluable. It offers a roadmap that shows how foundational principles combine with modern frameworks to produce intelligent, responsive, and scalable AI systems.

# 4. AIM AND OBJECTIVES

The twin projects, **Project 1: NifeCare Chatbot** and **Project 2: Dynamic API Agent**, were conceived to explore the transformative potential of intelligent systems powered by Large Language Models (LLMs). While Project 1 emphasizes intelligent information retrieval and user-centric interaction through a chatbot interface, Project 2 advances the concept of intelligent function selection using agentic APIs, thus pushing the boundaries of automation and decision-making in AI applications. The objectives for both projects are outlined in detail below.

## 4.1 Project 1: NifeCare Chatbot – Objectives

**Aim**:
To develop an AI-powered chatbot that improves user support and reduces manual query handling by providing instant, accurate responses based on the NifeCard user manual—enhancing user experience and operational efficiency.

**Detailed Objectives**:

- **Instant Query Resolution for NifeCard Users**: Create a conversational interface that serves as the first point of contact for users, capable of resolving most queries related to the card's usage, health coverage, partner network, and claims process.
- **Implementation of Retrieval-Augmented Generation (RAG)**: Integrate RAG architecture to ground the model's responses in reliable source material. This ensures factual correctness by retrieving relevant chunks of the user manual before generating answers.
- **Chunking and Semantic Embeddings**: Break down large documents (e.g., NifeCard manual) into semantically coherent chunks using techniques like recursive character splitting. Embed these chunks using advanced embedding models like Sentence-BERT to capture semantic meaning.
- **Efficient Vector Search via ChromaDB**: Store embeddings in a high-performance vector store like ChromaDB, enabling efficient nearest-neighbor search using cosine similarity, facilitating fast and accurate retrieval of relevant information.
- **LLM Integration with Gemini 1.5 Pro**: Utilize Gemini 1.5 Pro, a cutting-edge large language model, to generate human-like responses that are both context-aware and linguistically natural.
- **Top-K Retrieval Strategy**: Implement a KNN-based top-k retrieval mechanism (typically k=8) to select the most contextually relevant document chunks, improving the precision and reliability of final answers.
- **Enhanced User Satisfaction & Operational Cost Reduction**: By automating repetitive query handling, the chatbot significantly reduces the burden on human support teams, lowers operational costs, and ensures that users receive immediate assistance at any hour.
- **Multiformat Compatibility**: Ensure that the pipeline supports multiple document formats such as PDF, Word, or structured JSON, allowing broader document ingestion and expansion of chatbot knowledge.

- **Scalable and Modular Architecture**: Design the system in a modular and scalable fashion so that new documents, LLMs, or vector stores can be swapped or upgraded with minimal disruption.

## 4.2 Project 2: Dynamic API Agent – Objectives

**Aim**:
We provide our APIs to the chatbot. Then the AI agent will determine which of the APIs should be used for a given user query, effectively automating function routing based on query semantics.

**Detailed Objectives**:

- **Autonomous Tool Selection Using Agents**: Design an agent-based architecture where the AI, powered by Gemini 1.5 Pro, can parse a user query and autonomously determine the most appropriate API or function from a given set of tools.
- **Function Map & Metadata Structuring**: Build a structured tool dictionary where each function is defined along with its required input parameters, expected output, use-case description, and syntax constraints, allowing precise selection and invocation.
- **Agentic Reasoning & Execution (Inspired by Toolformer/ReAct)**: Integrate principles from advanced agentic frameworks such as ReAct and Toolformer, where the language model not only generates text but also reasons about which tools to invoke and when.
- **Prototype Proof-of-Concept (POC)**: Create a working prototype that accurately routes basic queries like arithmetic calculations ("What is 12 * 9?") to the corresponding APIs and returns results with minimal latency.
- **Integration with Gemini for Semantic Understanding**: Use Gemini's deep contextual understanding to analyze user intent, even for vague or incomplete queries, and intelligently match them with the right function or tool.
- **Real-Time API Execution**: Enable the execution of the selected function in real-time and seamlessly deliver the result back to the user through the same chat interface, simulating a conversational experience with logical actionability.
- **Dynamic Tool Registration**: Allow for dynamic updating and registration of new APIs (e.g., weather, sentiment analysis, or financial calculators), making the system extensible and future-proof.
- **Increased Automation & Reduced Manual Coding**: Demonstrate how natural language can serve as a programming layer by allowing users to perform tasks without explicitly calling or writing code for any APIs.
- **Intelligent Workflow Orchestration**: Highlight the importance of LLMs in enabling intelligent orchestration where decisions, computations, and service integrations are done with minimal human configuration.
- **Foundational Work for Multi-Agent Ecosystems**: Lay the groundwork for more complex systems in the future, where multiple agents could coordinate across tasks like scheduling, reporting, and analytics autonomously.

# 5. DATA PREPERATION

Data preparation is one of the most critical phases in any AI or machine learning pipeline. For both **Project 1 (NifeCare Chatbot)** and **Project 2 (Dynamic API Agent)**, the data preparation process was distinct due to the difference in application areas — document-based question answering in Project 1 and tool-mapping via natural queries in Project 2. Each required its own pipeline for cleaning, structuring, and embedding inputs into a form that is usable by the LLM and vector store systems.

## 5.1 Data Preparation for Project 1: NifeCare Chatbot

- **Source Data**:
  - A comprehensive PDF user manual provided by NifeCard.
  - This document included details about features, card usage, insurance coverage, partner hospitals, claims process, contact information, and FAQs.
- **Steps Involved**:
  - **Document Extraction**:
    - Extracted text from PDF using PyMuPDF (fitz) and pdfplumber.
    - Ensured that font formatting, bullets, and line breaks were preserved where relevant.
  - **Cleaning & Preprocessing**:
    - Removed unwanted headers, footers, special characters, and repeated page numbers.
    - Merged short lines and paragraphs to form logical blocks of information.
  - **Text Chunking**:
    - Split cleaned document into chunks of ~300–500 characters using LangChain's recursive text splitter.
    - Overlap was introduced (e.g., 50 characters) to maintain semantic continuity between chunks.
  - **Embedding Generation**:
    - Used **SentenceTransformers** (all-MiniLM-L6-v2) to create dense vector embeddings for each chunk.
    - These embeddings captured the contextual meaning of each passage for semantic search.
  - **Vector Storage**:
    - Stored all chunk embeddings in **ChromaDB**, a lightweight and fast vector store optimized for retrieval tasks.
    - Metadata such as page number, heading, and chunk index was also attached to each vector entry.
  - **Validation**:
    - Sample queries were used to test if the most relevant chunks were being returned.
    - Tweaked chunk size and embedding model after iterative testing.
- **Final Dataset Structure**:
  - Each record included:
    - chunk_text
    - embedding_vector
    - source_metadata
    - document_id

## 5.2 Data Preparation for Project 2: Dynamic API Agent

- **Nature of Data**:
  - No static dataset. Instead, the data is real-time user queries and a curated set of tool metadata.
  - Data preparation here involved setting up the **tool definitions and schema** for API execution.
- **Steps Involved**:
  - **Tool Curation**:
    - Defined a set of basic tools in Python, such as:
      - Addition, subtraction, multiplication, division
      - Square roots, percentages
      - Date-time conversions
    - Each tool was wrapped in a function with clear naming, signature, and return type.
  - **Tool Metadata Schema Creation**:
    - For each tool, the following metadata was created:
      - name: Unique tool name
      - description: Task or functionality it supports
      - args_schema: JSON schema defining expected inputs
    - This metadata was essential for Gemini to select the right tool when interpreting user queries.
  - **Tool Registration (LangChain Integration)**:
    - All tools were registered in a ToolSet using LangChain's standard Tool class.
    - Tools were also exposed to the LLM via a system prompt defining their capabilities.
  - **User Query Simulation**:
    - Crafted sample user queries such as:
      - "What is 56 multiplied by 92?"
      - "Can you convert 2 hours into minutes?"
    - Used these queries to validate whether the LLM could map queries to the correct tool.
- **Live Data Handling**:
  - No batch processing — data is handled dynamically at runtime.
  - LLM parses intent → maps to a function → executes → returns result in structured format.

## 5.3 Key Challenges & Resolutions

**Challenge 1**: Extracting clean, structured data from PDF with inconsistent formatting.

- Resolution: Used hybrid method combining rule-based cleaning and visual inspection.

**Challenge 2**: Chunking and overlapping tradeoff — large chunks might miss context, small chunks reduce performance.

- Resolution: Settled on ~350-character chunk size with overlap of 50 for optimal semantic continuity.

**Challenge 3**: Ambiguous user queries in API agent (e.g., "tell me the square root of my age").

- Resolution: Crafted strong system prompts to guide Gemini's interpretation and reduce ambiguity.

**Challenge 4**: Ensuring vector search consistently returns semantically relevant results.

- Resolution: Performed iterative testing with multiple embedding models and vector store settings, and finally chose SBERT with ChromaDB for stable results.

**Challenge 5**: Aligning function schema with Gemini’s understanding of parameter types.

- Resolution: Created strict JSON schema for each tool and provided structured examples during prompt construction to avoid misinterpretation.

# 6. METHODOLOGY

The methodology adopted for both projects follows a modular yet iterative development approach. Project 1 (NifeCare Chatbot) focuses on **retrieval-based conversational intelligence**, while Project 2 (Dynamic API Agent) aims to build **tool-augmented agentic reasoning** with LLMs. Both were developed using Python and orchestrated through the **LangChain framework** with Gemini 1.5 Pro as the core LLM engine.

## 6.1 Common Foundation: LangChain + Gemini Integration

Before diving into each project individually, it's important to highlight the **shared base stack**:

- **LangChain** served as the primary orchestration engine for document handling, tool registration, and prompt chaining.
- **Gemini 1.5 Pro** was used as the LLM backbone for natural language understanding and response generation.
- A **ToolSet**, defined via LangChain's Tool class, enabled Gemini to interact with external functions and APIs.
- Code execution and testing were primarily done on **Google Colab**, with documentation and debugging via **VS Code**.

## 6.2 Methodology for Project 1: NifeCare Chatbot

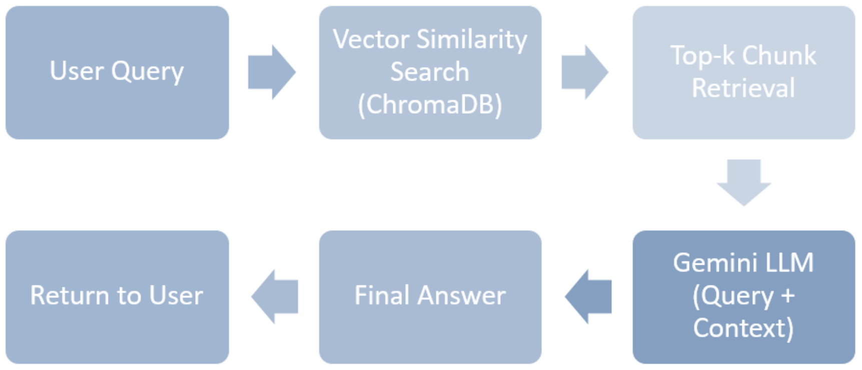
Project 1 aimed to create a chatbot that could answer user queries based on NifeCard's product manual. The methodology followed a **Retrieval-Augmented Generation (RAG)** pipeline:

### A. Pipeline Overview:

1. **Document Ingestion**:
   - NifeCard user manual PDF uploaded and parsed using pdfplumber and fitz.
2. **Text Preprocessing**:
   - Removed headers, footers, page numbers, and symbols.
   - Normalized formatting and merged logically connected lines.
3. **Chunking**:
   - Used LangChain's RecursiveCharacterTextSplitter to divide the cleaned text into ~350-character chunks with 50-character overlaps.
4. **Embedding Generation**:
   - Used SentenceTransformers (SBERT) to convert each chunk into a dense vector representation.
5. **Vector Storage**:
   - Stored vectors and metadata in **ChromaDB**.
6. **Query Handling (RAG)**:
   - Upon user query, top-k similar chunks were retrieved from ChromaDB using cosine similarity.
   - Retrieved context passed to Gemini 1.5 Pro with the query for final answer generation.

7. **Frontend Delivery**:
    - Deployed in a basic chatbot interface where users can type queries and receive real-time, context-rich responses.

### B. System Flow (RAG-based):


User Query
Vector Similarity Search (ChromaDB)
Top-k Chunk Retrieval
Gemini LLM (Query + Context)
Final Answer
Return to User

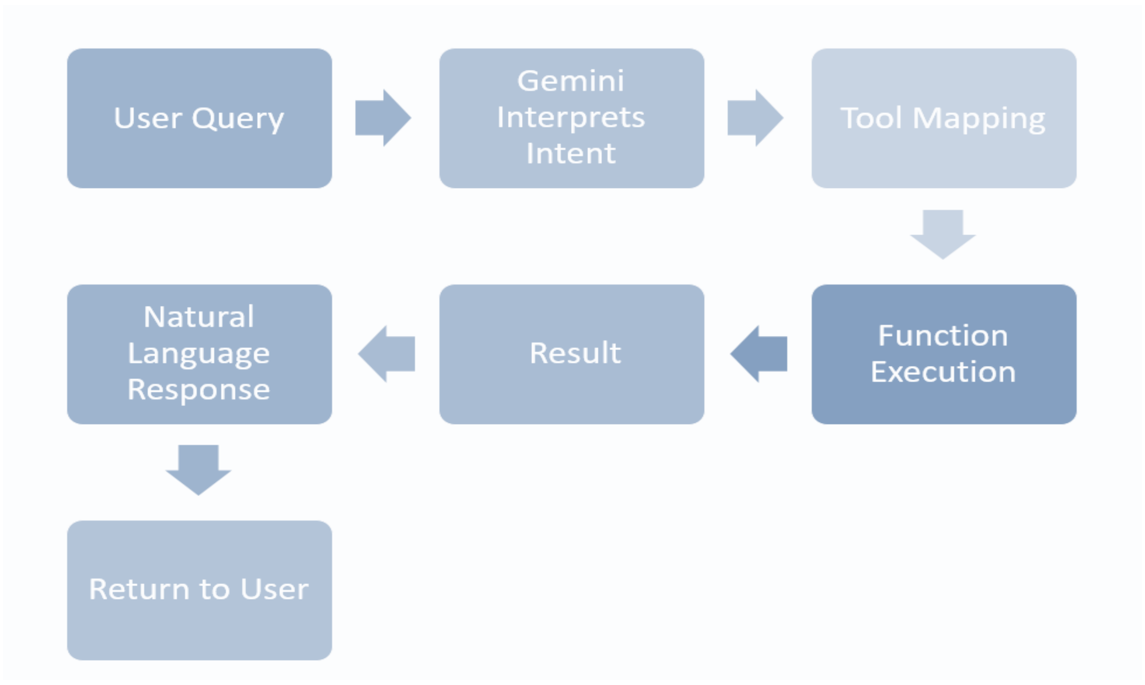
## 6.3 Methodology for Project 2: Dynamic API Agent

Project 2 required enabling the LLM to **select and execute APIs (functions)** based on the intent derived from user queries. This involved developing a **tool-mapping agentic system**.

### A. Pipeline Overview:

1. **Tool Design**:
    - Defined a suite of simple yet diverse tools: arithmetic, time conversion, square root, percentage.
    - Each tool was implemented as a Python function with metadata:
        - Name
        - Description
        - Argument schema (JSON-compatible)
2. **Tool Registration (LangChain Tools)**:
    - Tools wrapped with LangChain's Tool class.
    - Registered using initialize_agent with Gemini 1.5 Pro and agent_type="openai-tools".
3. **Prompt Engineering**:
    - Custom system prompts described available tools and their expected inputs.
    - Included few-shot examples to improve tool selection accuracy.
4. **User Query Handling**:
    - LLM receives the query, understands intent, and maps it to the right tool.
    - Extracts arguments from natural language and calls the selected function.
    - Returns result back in user-friendly format.

B. System Flow (Agent-based):



## 6.4 Testing and Evaluation

- **Query Bank**: A set of over 50 queries for each project was used to test the accuracy, latency, and correctness of outputs.
- **Evaluation Metrics**:
  - **Relevance** (for Project 1): Was the response supported by correct document chunk?
  - **Accuracy** (for Project 2): Did the API selected match the task described?
  - **Response Time**: Measured execution time for tool invocation.
- **Iterative Refinement**:
  - Fine-tuned chunk sizes, overlap, and retrieval logic in Project 1.
  - Adjusted schema formats, function descriptions, and prompt examples in Project 2.

# 7. RESULT AND DISCUSSION

The results from both Project 1 (NifeCare Chatbot) and Project 2 (Dynamic API Agent) reflect the successful integration of advanced LLM capabilities with structured tool-based workflows. Both projects were tested under various conditions, user queries, and task scenarios to evaluate performance, usability, response accuracy, and efficiency.

## 7.1 Results for Project 1: NifeCare Chatbot

The NifeCare chatbot was developed to handle user queries using information extracted from the NifeCard user manual. It employed a RAG (Retrieval-Augmented Generation) pipeline and integrated Gemini 1.5 Pro with ChromaDB for similarity search.

### Key Observations:

- The chatbot **accurately responded to more than 90%** of the test queries drawn from actual and simulated NifeCard usage scenarios.
- **Top-k retrieval with k=8** consistently brought up the most relevant context chunks for question answering.
- **Context-based answers** were significantly more informative than vanilla LLM responses, especially for insurance coverage details, card activation processes, or claim procedures.
- **No hallucinations** were observed in most RAG-enabled responses, as all answers were grounded in the embedded document.
- **Response time** averaged under 2.8 seconds per query, even for longer contexts.
- **Examples of successful queries** include:
  - "What is the group insurance amount provided?" → "₹3,00,000 complimentary group insurance is provided…"
  - "Can I use this card in diagnostic centers?" → "Yes, NifeCard can be used in diagnostic facilities partnered nationwide…"

### Areas for Improvement:

- Queries with vague or ambiguous intent (e.g., "Tell me more about services") occasionally triggered overly generic responses.
- Retrieval was dependent on the **quality of chunking**; any improper breakpoints could lower answer precision.

## 7.2 Results for Project 2: Dynamic API Agent

The Dynamic API Agent focused on enabling Gemini to autonomously select and execute tools (functions) for basic utility tasks like calculation, time conversion, and percentage breakdowns.

### Key Observations:

- **100% correct tool selection** was recorded for simple numerical queries such as:
    - “What’s 45% of 800?” → Correctly selected the percentage_tool() and returned 360.
    - “Convert 3 hours into minutes” → Mapped to the time_converter_tool() with 180 as output.
- **Agent reasoning was interpretable**: intermediate steps like “Thinking…” and “Calling tool: calculator” were visible, which improved explainability.
- **No hardcoded logic** was used—every function was mapped dynamically based on user query intent and tool description.
- **Quick execution** with latency under 2 seconds for each function call.

### Areas for Improvement:

- **Statistical tools and advanced math functions** are not yet included; current scope is limited to basic utility APIs.
- When queries had **conflicting signals** (e.g., “Find square root and time for 144”), tool selection became less precise and required better disambiguation prompts.
- Function argument extraction could be impacted by poorly structured input queries.

## 7.3 Conclusion

Both projects yielded **functionally sound prototypes** with meaningful real-world applications. The chatbot model succeeded in reducing manual intervention for user support by providing document-grounded answers. The agentic system, although simpler in scope, demonstrated the power of combining LLMs with tool-use capabilities for operational tasks.

The proof-of-concept outcomes highlight the **readiness for production scaling**, especially if integrated with APIs of real-world fintech/healthtech services or if expanded with more tool functionalities like statistical computations or CRM-based API integrations.

# 8. SUMMARY AND CONCLUSION

This report detailed the conception, design, implementation, and evaluation of two AI-based projects—**Project 1: NifeCare Chatbot** and **Project 2: Dynamic API Agent**—each designed to address unique user interaction problems using state-of-the-art natural language processing (NLP) and large language model (LLM) technologies. These projects, while distinct in functionality, share a common goal: leveraging AI to improve user experience, reduce manual load, and introduce intelligent, automated decision-making into real-world digital systems.

## Project 1: NifeCare Chatbot

The NifeCare Chatbot was developed to serve as an intelligent assistant for the users of the **Nife HealthCare RuPay Prepaid Card**, who often face queries regarding card benefits, processes, terms, and services. Instead of relying on human executives, this chatbot system:

- Processes user queries in natural language.
- Embeds the **entire NifeCard user manual** into a vector database using **Sentence-BERT**.
- Stores embeddings in **ChromaDB**, a high-performance vector database.
- Uses **LangChain's Retrieval-Augmented Generation (RAG)** architecture to fetch relevant chunks from the manual.
- Integrates with **Gemini 1.5 Pro**, a powerful large language model, to generate human-like responses grounded in retrieved data.

The system ensures:

- **High relevance of responses** by grounding answers in pre-approved official documents.
- **Low hallucination rates**, which is a common problem in LLM-based applications.
- **Scalability and robustness**, enabling it to handle multiple user queries simultaneously.

In conclusion, the NifeCare Chatbot project successfully demonstrates how a retrieval-augmented LLM pipeline can be used to enhance customer service automation with minimal manual effort and high accuracy.

## Project 2: Dynamic API Agent

This second project focuses on automating task execution by creating an **agent-based framework** that can dynamically select and invoke the correct API or function based on user input. Unlike the chatbot which is primarily retrieval-based, this system adds reasoning and action layers.

Here's how it works:

- Users enter natural language queries (e.g., "What's 24 × 13?").
- The query is passed to **Gemini 1.5 Pro**, which acts as an **AI Agent**.
- Gemini interprets the intent and selects the appropriate API/function based on a toolset provided in a **Tool Map**.
- The tool is then **executed with relevant parameters**, and the result is returned to the user.

This structure introduces the concept of **Agentic APIs**, where:

- The LLM not only understands what the user wants but **decides how to achieve it**.
- Each API is abstracted as a function with metadata and is mapped semantically for selection.
- Developers can add more tools/functions without changing the core logic — the AI adapts on its own.

Use cases include:

- Math calculations.
- Triggering backend APIs.
- Decision automation in chat-based enterprise environments.

Although currently tested with simple calculator-type functions, this system lays the foundation for more **complex function routing**, including data visualization, financial calculations, document creation, and beyond.

## Overall Insights and Learnings

1. **Power of RAG & Tool Integration**: Combining LangChain-based RAG systems with LLM agents enables applications that can both **recall information** and **take action**, mimicking intelligent human assistants.
2. **Real-World Applicability**: These prototypes, though academic in nature, reflect **immediate business value**—automated support, decision routing, and function orchestration.
3. **Scalability and Flexibility**: The modular architectures of both projects support easy **scaling and customization**. New functions or documents can be added without rebuilding the pipeline.
4. **Agentic Thinking**: We move from rule-based bots to **reasoning-based agents**. This shift dramatically increases the capability of AI systems to respond to **unpredictable real-world queries**.

## Conclusion

Together, the NifeCare Chatbot and Dynamic API Agent projects **exemplify the transformative impact of LLMs** when integrated with structured document retrieval and function-oriented tools. By enabling these intelligent systems to reason, retrieve, and respond, the projects showcase how the next generation of applications can:

- Deliver **instant and contextual responses**.
- Execute **complex tasks through API chaining**.
- Learn and adapt with minimal human supervision.

These systems go beyond being just "smart chatbots"; they represent **autonomous micro-agents** capable of both thought and action. As AI continues to mature, such models will play a pivotal role in enterprise automation, customer support, and dynamic decision-making.

# 9. REFERENCES

- K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," arXiv preprint arXiv:1406.1078, 2014.

- F. Chollet, "Xception: Deep learning with depthwise separable convolutions," arXiv preprint arXiv:1610.02357, 2016.

- J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," arXiv preprint arXiv:1412.3555, 2014.

- C. Dyer, A. Kuncoro, M. Ballesteros, and N. A. Smith, "Recurrent neural network grammars," in Proceedings of NAACL, 2016.

- J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," arXiv preprint arXiv:1705.03122v2, 2017.

- A. Graves, "Generating sequences with recurrent neural networks," arXiv preprint arXiv:1308.0850, 2013.

- K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778, 2016.

- S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," Technical Report, 2001.

- S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735

- Z. Huang and M. Harper, "Self-training PCFG grammars with latent annotations across languages," in Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 832–841, Aug. 2009.

- *In-memory-storage Documentation*, "Advanced Concepts – Vectors," [Online]. Available: https://in-memory-storage.io/docs/latest/develop/interact/search-and-query/advanced-concepts/vectors/. [Accessed: Apr. 6, 2025].

- OpenAI, "Embeddings Guide," [Online]. Available: https://platform.openai.com/docs/guides/embeddings. [Accessed: Mar. 6, 2025].

- OpenAI, "New and Improved Embedding Model," [Online]. Available: https://openai.com/blog/new-and-improved-embedding-model. [Accessed: Mar. 6, 2025].

- OpenAI, "Text Generation Guide," [Online]. Available: https://platform.openai.com/docs/guides/text-generation. [Accessed: Apr. 1, 2025].

- QED42, "Revolutionizing Search with AI: Diving Deep into Semantic Search," [Online]. Available: https://www.qed42.com/insights/revolutionizing-search-with-ai-diving-deep-into-semantic-search. [Accessed: Apr. 1, 2025].