AIKO: AI-POWERED KNOWLEDGE ORGANIZER

B.Tech. PROJECT

Submitted to Rashtrasant Tukdoji Maharaj Nagpur University, Nagpur
in Partial Fulfillment of the
Requirements for the Degree of BACHELOR OF TECHNOLOGY in
COMPUTER SCIENCE AND ENGINEERING

By

Aditya S. Deshmukh (ID 2021016600840367)
Devansh S. Parapalli (ID 2021016600817392)
Kaustubh D. Warade (ID 2021016600880071)
Yashasvi B. Thool (ID 2021016600869734)

Guide
Dr. Devchand J. Chaudhari
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GOVERNMENT COLLEGE OF ENGINEERING NAGPUR

2024-2025

AIKO: AI-POWERED KNOWLEDGE ORGANIZER

B.Tech. PROJECT

Submitted to Rashtrasant Tukdoji Maharaj Nagpur University, Nagpur
in Partial Fulfillment of the
Requirements for the Degree of BACHELOR OF TECHNOLOGY in
COMPUTER SCIENCE AND ENGINEERING

By

Aditya S. Deshmukh (ID 2021016600840367)
Devansh S. Parapalli (ID 2021016600817392)
Kaustubh D. Warade (ID 2021016600880071)
Yashasvi B. Thool (ID 2021016600869734)

Guide
Dr. Devchand J. Chaudhari
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GOVERNMENT COLLEGE OF ENGINEERING NAGPUR

2024-2025

GOVERNMENT COLLEGE OF ENGINEERING NAGPUR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled, "AIKO: AI-POWERED KNOWLEDGE ORGANIZER" which is being submitted herewith for the award of B. Tech, is the result of the work completed by (1) Aditya S. Deshmukh (2) Devansh S. Parapalli (3) Kaustubh D. Warade (4) Yashasvi B. Thool under the guidance of Dr. Devchand J. Chaudhari.

(Dr. Devchand J. Chaudhari)	(Dr. Latesh G. Malik)	(Dr. Rewatkumar P. Borkar)
Guide	Head of Department	Principal

DECLARATION

We hereby declare that the project entitled, "AIKO: AI-POWERED KNOWLEDGE ORGANIZER" was carried out and written by us under the guidance of Dr. Devchand J. Chaudhari, Assistant Professor, Department of Computer Science and Engineering, Government College of Engineering, Nagpur. This work has not been previously formed the basis for the award of any degree or diploma or certificate nor has been submitted elsewhere for the award of any degree or diploma.

Date:

Place: Nagpur

(1) Aditya S. Deshmukh

University Enrollment Number: 2021016600840367

(2) Devansh S. Parapalli

University Enrollment Number: 2021016600817392

(3) Kaustubh D. Warade

University Enrollment Number: 2021016600880071

(4) Yashasvi B. Thool

University Enrollment Number: 2021016600869734

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to all those who have contributed to the successful completion of this project.

We are deeply indebted to our project guide, Dr. Devchand J. Chaudhari, for his exemplary mentorship and unwavering support throughout this endeavor. His profound knowledge, insightful guidance, and constructive feedback have been instrumental in shaping our research and overcoming technical challenges.

We extend our heartfelt appreciation to Dr. Latesh G. Malik, Head of the Department of Computer Science and Engineering at Government College of Engineering Nagpur, for providing us with the necessary resources and infrastructure.

Our sincere appreciation to Dr. Rewatkumar P. Borkar, Principal, Government College of Engineering Nagpur, for fostering an environment of academic excellence and innovation within our institution.

We are grateful to our peers who have contributed significantly to this project and the AIKO Test Users Group for their invaluable feedback and suggestions during AIKO's development. Their technical acumen and collaborative spirit have greatly enhanced the quality of our work.

We would also like to acknowledge the support of the entire Computer Science and Engineering department, whose faculty members have been a constant source of knowledge and inspiration throughout our academic journey.

ABSTRACT

AIKO (AI-powered Knowledge Organizer) is an innovative system designed to address the challenges of information overload and fragmented knowledge management in the digital age. Leveraging advanced artificial intelligence techniques, including natural language processing and machine learning, AIKO provides a comprehensive solution for integrating, processing, and retrieving information contained within diverse modalities. The main motivations for the project were the lack of a centralized personal knowledge management system capable of handling multi-modal data and the need for a more efficient and easy to use system for knowledge organization and retrieval.

AIKO has been developed using the latest technologies such as GenAI and large Language Models for content processing with the waterfall model of software development. The system features an innovative modular architecture, increasing resiliency and scalability. A new framework for language model integration has been developed as part of this project. A plugin architecture allows for easy expansion of data source connectors, enabling seamless integration with various platforms. AIKO was developed as a proof-of-concept to demonstrate the feasibility of a personalized knowledge management system. A core observation was the ability of AIKO to demonstrably improve the efficiency of knowledge management and retrieval tasks for users. The system's efficacy is further demonstrated through the results obtained from a user survey. Key results for AIKO include a 12-fold improvement in search and retrieval times along with a satisfaction score of 4.67 out of 5.

AIKO highlights the need to improve organizational efficacy and productivity by utilization of newer technologies. AIKO also has the potential to revolutionize the way individuals and organizations manage and access information, providing a more efficient and personalized knowledge management system. Processing of video and varied binary formats is a future direction for AIKO's vast processing capabilities.

CONTENTS

List of Figures

NOMENCLATURE

ACID Atomicity, Consistency, Isolation, Durability.
AES Advanced Encryption Standard.
AI Artificial Intelligence.
AIKO AI-powered Knowledge Organizer.
ANCE Approximate Nearest Neighbor Negative Contrastive Estimation.
API Application Programming Interface.
ASGI Asynchronous Server Gateway Interface.
BIM Binary Independence Model.
Blake2 A cryptographic hash function.
CD Continuous Delivery/Deployment.
CDN Content Delivery Network.
CDSSM Convolutional Deep Structured Semantic Model.
CI Continuous Integration.
CNN Convolutional Neural Network.
CSR Client-Side Rendering.
CSRF Cross-Site Request Forgery.
CSS Cascading Style Sheets.
CUDA Compute Unified Device Architecture.
DCP Distributed Computing Protocol.
DIP Dependency Inversion Principle.
DIS Data Ingestion Service.
DOM Document Object Model.
DPR Dense Passage Retrieval.
DRMM Deep Relevance Matching Model.
DSSM Deep Structured Semantic Model.
DUET Dual Embedding Text Matching.
EC2 Elastic Compute Cloud (Amazon EC2).
ECC Elliptic Curve Cryptography.
ES Module ECMAScript Module.
GPU Graphics Processing Unit.
HMM Hidden Markov Model.
HMR Hot Module Replacement.
HTTP Hypertext Transfer Protocol.
IAM Identity and Access Management.
IDE Integrated Development Environment.
IPS Information Processing Service.
IR Information Retrieval.
JAX Just After Execution (a Python library for machine learning).
JSON JavaScript Object Notation.
JWT JSON Web Token.
K-NRM Kernel-based Neural Ranking Model.
LLM Large Language Model.
LSA Latent Semantic Analysis.
LSP Liskov Substitution Principle.

MBps Megabytes per second.
MFA Multi-Factor Authentication.
ML Machine Learning.
NER Named Entity Recognition.
NLP Natural Language Processing.
OAuth Open Authorization.
ONNX Open Neural Network Exchange.
PCA Principal Component Analysis.
PDF Portable Document Format.
R2 Cloudflare R2 (object storage service).
RAM Random Access Memory.
RDBMS Relational Database Management System.
REST Representational State Transfer.
RNN Recurrent Neural Network.
RSA Rivest-Shamir-Adleman (a public-key cryptosystem).
S3 Simple Storage Service (Amazon S3).
SAML Security Assertion Markup Language.
SDD Software Design Document.
SEO Search Engine Optimization.
SRP Single Responsibility Principle.
SRS Software Requirements Specification.
SSD Solid State Drive.
SSG Static Site Generator.
SSR Server-Side Rendering.
t-SNE t-Distributed Stochastic Neighbor Embedding.
TorchScript A way to create serializable and optimizable models from PyTorch code.
UI User Interface.
UMAP Uniform Manifold Approximation and Projection.
vRAM Video Random Access Memory.
VSM Vector Space Model.
WSGI Web Server Gateway Interface.
XAI Explainable AI.

CHAPTER 1
INTRODUCTION

1.1 Problem Statement

The exponential increase in digital information has resulted in a growing challenge for individuals and organizations to manage, access, and utilize data efficiently. The issue manifests in two major forms: information overload and fragmented knowledge management. The overwhelming volume of data across multiple platforms, including emails, cloud storage, and social media, burdens users, making it difficult to prioritize and retain important information. Newer information often overshadows previous data, leading to a cognitive overload.

In addition, knowledge is fragmented, scattered across disparate platforms, devices, and formats, leading to inefficiencies in retrieval. Users experience delays in locating relevant information, as search mechanisms struggle with unstructured data, often resulting in reduced relevance and accuracy in search results. Current systems lack adequate contextual understanding and fail to process or link pieces of information cohesively. The lack of personalization further exacerbates the problem, as users' unique needs and preferences in organizing and accessing information are insufficiently catered to by existing systems.

These factors collectively lead to reduced productivity, increased cognitive stress, and missed opportunities for leveraging knowledge effectively.

1.2 Objectives

The AIKO project aimed to address the challenges highlighted with the following primary objectives.

- **Data Integration and Normalization**

The project's design called for implementing a unified data layer capable of handling heterogeneous data sources, ensuring seamless integration of information from various platforms. Development of robust APIs for efficient data ingestion was planned, allowing accommodation of diverse data formats and sources. A centralized

hub for consolidated information storage and management was envisioned, providing a comprehensive solution for organizations dealing with fragmented data ecosystems.

- **Advanced Search and Retrieval Mechanisms**

Development of a high-performance search engine optimized for large-scale data retrieval was a core objective. Implementation of advanced indexing techniques and Natural Language Processing (NLP) algorithms was planned to enable sophisticated semantic search capabilities. The focus on efficient cross-platform information retrieval aimed to significantly enhance data accessibility, allowing users to quickly locate and utilize relevant information across diverse data sources and formats.

- **Security Implementation**

Robust security measures were prioritized through the design of a multi-layered security architecture. Plans included implementation of state-of-the-art encryption protocols for data at rest and in transit, ensuring comprehensive data protection. Integration of OAuth 2.0 and SAML for authentication and authorization was intended, providing a secure framework for user access and data handling that met modern cybersecurity standards.

- **Automated Synchronization**

Addressing the challenges of maintaining data consistency across multiple platforms, the project aimed to design a sophisticated publish-subscribe system facilitating real-time updates. AIKO's system was to be complemented by advanced conflict resolution algorithms capable of managing multi-source data updates efficiently. The overarching goal was to ensure automatic data consistency maintenance across diverse platforms and devices, minimizing data discrepancies and enhancing overall system reliability.

- **Intuitive User Experience**

An exceptional user experience was targeted through the creation of a cross-platform accessible frontend interface. Plans included implementation of collaborative filtering algorithms and user preference customization features, allowing for a highly personalized interaction with the system. The focus on enhancing user engagement through

tailored interfaces and information feeds sought to optimize the efficiency and effectiveness of knowledge management processes for individual users and organizations alike.

1.2.1 Expected Outcomes

The implementation of AIKO was expected to yield several significant outcomes, including:

- **Enhanced Information Management**

Users were expected to experience a significant reduction in the time required to locate and retrieve information, with an expected improvement of 60% in efficacy. The system was expected to decrease the cognitive load on users through automated organization, allowing them to focus on more critical tasks rather than information management.

- **Operational Efficiency**

Users were expected to benefit significantly from the system's streamlined data ingestion and retrieval processes. AIKO's deduplication and organizational capabilities were expected to improve collaborative workflows as well as accelerate decision-making processes.

- **System Performance**

AIKO was expected to maintain high availability and scalability to support the constantly growing needs of the users. The system's real-time capabilities were expected to ensure data consistency across multiple devices and platforms.

- **Security & Privacy**

AIKO was expected to provide robust security features, ensuring data privacy and integrity both at rest and in transit. Access control and threat detection mechanisms were expected to safeguard sensitive information from unauthorized access or data breaches.

1.3 Organization Of Report

The report is structured to provide a comprehensive overview of the AIKO project, detailing the theoretical foundations, system design and architecture, implementation methodology, project management aspects and future research directions.

Chapter 1 provides a detailed introduction to the project, outlining the problem statement, relevance of AIKO, initial goals, and the report's structure.

Chapter 2 dives into the review of existing knowledge management systems, highlighting the features, limitations, and relevance to AIKO's development.

Chapter 3 delves into the theoretical framework underpinning AIKO, discussing the aspects behind AIKO's design and architecture, implementation methodology, and project management.

Chapter 4 provides an in-depth analysis of the implementation methodology adopted for AIKO, outlining the system design, architectural decisions, development process, tools, technologies, and frameworks utilized.

Chapter 5 discusses AIKO's user interface design and the results obtained from AIKO's development and implementation.

Chapter 6 evaluates the project goals, analyzes AIKO's strengths and limitations, and suggests future research and development directions.

Subsequent Appendices provide additional information, including links to online versions of the project and its documentation, system prompts used for LLMs and the database schema used in AIKO.

The report is concluded with the attachment of the project's publications.

CHAPTER 2
REVIEW OF LITERATURE

2.1 Overview of Existing Systems

This section provides a comprehensive analysis of existing knowledge management systems, highlighting their features, limitations, and relevance to AIKO's development.

Mem.ai [1] offered a platform focused on content organization and generation, but significantly limited its capabilities by operating exclusively within its proprietary ecosystem. The platform lacked support for importing external documents, which restricted its versatility for broader information management purposes. While the closed environment potentially allowed for more robust integration of its core features, it presented a major drawback for users needing to work with external data sources. The AIKO project team identified this lack of external document support as a critical limitation for developing a comprehensive knowledge management system that could serve diverse user needs.

Brain Assistant [2] presented a browser-centric solution utilizing open-source models. While it supported file processing, it was confined to text-based documents and required manual file uploads, which created additional friction for users working with diverse data types. Its browser-only accessibility reduced usability in scenarios requiring offline access or alternative interaction modes. Brain Assistant's limited file type support and absence of multimedia content capabilities diminished its versatility compared to other solutions. The AIKO project team determined that this lack of multimodal support represented a significant drawback for the Indian ecosystem, where users frequently interact with both printed-scanned materials and digital content [3].

AI Brain Bank [4] was a simple platform focused on content retrieval through the use of open-source models for content embedding. It lacked features such as content generation and support for additional communication channels, which made it more suited to straightforward retrieval tasks. The minimalistic design

may have appealed to users seeking ease of use, but it did not address more complex needs like multi-modal information processing or customizable workflows. The AIKO project team identified the lack of customization options (even while AI Brain Bank uses open-source models, whose tooling provides for a decent level of abstractions [5]) and the absence of multi-modal support as key limitations that would hinder its adoption in a diverse knowledge management environment.

iWeaver [6] was primarily concerned with website link analysis and used Open-Graph for generating similarity scores between links. However, it restricted itself to working exclusively with web links and provided limited functionality for processing audio or video content. Paid options were available for video and audio processing, but the use of public models without user-selectable options restricted the customisation potential. Its reliance on OpenGraph made it a niche tool, primarily useful for users focused on web-based content analysis. The lack of diverse content support and customisation options limited its applicability in broader knowledge management scenarios.

Keepi.ai [7] offered more versatility by supporting a range of content types, including URLs, text documents, and images. However, it fell short in supporting video and audio, which made it less suitable for environments requiring multi-modal content processing. Despite its broader range of file support compared to other platforms, its limitations in handling multimedia content indicated it was not a comprehensive solution for diverse knowledge management needs. The absence of audio and video support constrained its usability in projects demanding a more complete information capture mechanism. The AIKO project team set the ability to process text, pdf and .docx as a benchmark for the minimum level of content support required for a comprehensive knowledge management system, aiming to surpass the benchmark standard by providing a more versatile and inclusive platform.

2.2 Review of Existing Literature

The development of advanced knowledge management systems capable of handling diverse data formats has gained significant momentum following the publication of the seminal paper “Attention is All You Need” [8], which revolutionized natural

language processing. Subsequent advancements in Large Language Models (LLMs) and semantic search technologies have substantially improved information retrieval capabilities. This review synthesizes relevant research, examining contemporary approaches to information retrieval and knowledge augmentation that inform AIKO's development.

Recent advancements such as *AssistRAG* enhance LLMs with retrieval-augmented generation (RAG) capablities by employing intelligent information assistants [9]. The paper also describes the pitfalls and shortcomings of the earliest "Retrieve-Read" techniques, prompt-based RAG techniques and Supervised Fine-Tuning (SFT) methods. To cope with these challenges, AssistRAG proposes an Assistant-based Retrieval Augmented Generation, which integrates newer use cases like tool-use to improve upon the previous techniques. Similarly, the *ZeroG* knowledge engine proposes a two model system to ground retrieval in validated sources. [10]. A case study on biodiversity publications demonstrated improved search relevance when integration LLMs with structured indexing methods [11].

Efficient information retrieval relies heavily on embedding representations and vector databases. Research on nearest neighbor search (NNS) in high-dimension spaces reveals that NNS is resilient to the "curse of dimensionality" [12]. Additionally, the paper explains the irrelevance of choice of the distance function and proposes methods for further optimization of dense vector-related applications. Furthermore, LLM-powered query generation along with NNS over the available tools allows for better retrieval statistics [13].

Challenges in ensuring factual accuracy and usability in search systems have been identified [14, 15]. While these insights are tangential to our primary focus, they highlight user-centric considerations crucial for adoption. Similarly, grounding hypothesis generation in reliable knowledge remains an ongoing challenge for all systems leveraging LLMs for information retrieval [16].

Existing research provides a robust foundation for the development of a modular, AI-powered knowledge organizer. By integrating insights

from semantic search, vector database efficiency, and modularity, AIKO could address the unique challenges of managing diverse data formats.

2.3 Gaps in Current Solutions

The review of existing systems and research revealed several common limitations that hindered their effectiveness in addressing the challenges of information overload and fragmented knowledge management. A non exhaustive list of key gaps identified in current solutions is given in the following sub-sections.

2.3.1 Limited Multi-Modal Integration

Existing systems often focus on text-based content processing, neglecting the importance of multi-modal integration. The lack of support for audio, video, and image data types restricts the comprehensive capture and retrieval of information across diverse media formats. The limitation results in incomplete knowledge management solutions that fail to cater to the varied content needs of users. Most pipelines are able to process unstructured text very efficiently.

2.3.2 Subpar Search Functionalities

The main motive for AIKO was the limitations in search [17] provided by popular cloud storage platforms, such as Google Drive and Dropbox. These platforms provide basic search functionalities that are not context-aware and do not support semantic search. The search was so basic that it lacked introspection into the content of the files, searching only by filename. The lack of advanced indexing mechanisms and semantic understanding hampers the accuracy and relevance of search results, leading to inefficiencies in information retrieval. The AIKO project team recognized the need for a more sophisticated search engine that could deliver context-aware and cross-modal search capabilities to enhance the user experience and improve knowledge management efficiency.

2.3.3 Fragmented Data Management

The target demographic for AIKO struggles with data management due to information fragmentation across multiple platforms, including emails (often self-emails), cloud storage services, and social media channels (self-

groups or personal messages). This fragmentation creates significant cognitive overhead as users must track data locations manually across disparate systems.

The lack of a unified data management system results in inefficient retrieval processes and disorganized information, contributing to cognitive overload and reduced productivity. The ZeroG knowledge engine approach [18] demonstrates how integrated systems can improve information coherence across sources. AIKO builds upon these concepts to develop a comprehensive knowledge management system that integrates and normalizes data from diverse sources, providing users with a centralized platform for efficient information retrieval and management.

2.3.4 Inadequate Personalization

Differences in user's data ingest and processing requirements were not adequately addressed by existing systems. The models in use have differing abilities [19], and the lack of customisation options for model choice and hyperparameters restricted the adaptability of the systems. The absence of personalized interfaces and user-centric design features limited the usability of knowledge management systems, as users were unable to tailor the platforms to their specific needs and preferences. The AIKO project team recognized the importance of user empowerment and aimed to provide a highly customizable and personalized system that could adapt to individual user requirements, enhancing the overall user experience and productivity.

2.4 Summary

The comprehensive analysis of existing knowledge management systems and relevant literature reveals several critical challenges that AIKO addresses through its innovative design and implementation.

2.4.1 Information Overload and Cognitive Burden

As highlighted, users face overwhelming volumes of data generated daily across various platforms. Current solutions either fail to address such caveats comprehensively or provide incomplete mechanisms for managing and filtering vast quantities of information. AIKO integrates advanced

AI techniques like NLP to prioritize, categorize, and synthesize information from multiple sources, significantly reducing the cognitive load on users.

2.4.2 Fragmented Data Across Platforms

The scattered nature of data across numerous platforms (emails, social media, cloud storage) results in inefficient retrieval processes. While existing solutions offer limited platform integration, AIKO's unified knowledge repository aggregates data from heterogeneous sources into a centralized hub, allowing users to access all necessary information from a single interface.

2.4.3 Inefficiencies in Information Retrieval

Current tools fall short in effectively searching through unstructured and multi-modal content types (text, audio, video). AIKO's modular architecture and use of machine learning addressed the shortcomings by enabling cross-platform, context-aware information retrieval. Its semantic search functionality ensures more accurate and faster query results.

2.4.4 Lack of Personalization

Users have diverse needs and preferences when it comes to how such users organize and access information. Current solutions rarely offer customization. AIKO introduces personalized interfaces and collaborative filtering mechanisms to tailor the knowledge organization process, catering to individual user requirements and improving overall engagement.

2.4.5 Security Concerns

Many existing platforms neglect to prioritize security or transparency. AIKO, however, incorporates multi-layered encryption and modern authentication protocols (OAuth 2.0 [20], SAML) to safeguard sensitive data. Furthermore, its focus on explainable AI (XAI) ensures transparency, allowing users to understand how their data is processed and managed – addressing key concerns highlighted regarding user trust in AI systems [14].

2.4.6 Inadequate Contextual Understanding

Existing systems often demonstrate limited ability to understand context within and across different information streams. Recent advances in LLMs [16] show how contextual understanding can be significantly improved through advanced model architectures. AIKO integrates these technologies to ensure retrieved information is not only relevant but also coherent in relation to other available data, providing users with a more comprehensive understanding of their knowledge base.

CHAPTER 3
THEORETICAL FRAMEWORK

Chapter 3 is intended to be an introduction to the framework of the technologies used in AIKO. The chapter covers a brief theoretical overview of the technologies used in the project, including Natural Language Processing (NLP), Machine Learning (ML), and Information Retrieval (IR). The explanations given here are not exhaustive, for more in-depth information, detailed mathematical proofs, and the latest advancements in the field, readers are encouraged to consult peer-reviewed papers, textbooks on Natural Language Processing and Machine Learning, and reputable online resources such as arXiv, academic journals, and technical blogs from leading AI research institutions.

3.1 Large Language Models

The following section provides a brief overview of Large Language Models, enough to understand the context of AIKO's development.

3.1.1 Fundamental Architecture and Principles

Large Language Models (LLMs) [8] represent a significant advancement in natural language processing [21] and artificial intelligence. These models are built upon the foundation of neural networks, specifically utilizing the transformer architecture introduced by Vaswani et al. (2017).

- **Transformer Architecture**

The transformer architecture forms the backbone of modern LLMs. It employs a self-attention mechanism that allows the model to weigh the importance of different parts of the input when processing each element. The architecture consists of several key components:

- **Multi-head attention layers**: These layers enable the model to focus on different aspects of the input simultaneously, capturing complex relationships within the data.
- **Feed-forward neural networks**: These networks process the output of the attention layers, allowing for non-linear transformations of the data.

- **Layer normalization**: The technique helps stabilize the learning process by normalizing the inputs to each layer.
- **Residual connections**: These connections facilitate the flow of information across the network, mitigating the vanishing gradient problem in deep networks.

Figure 3.1: Transformer Architecture

- **Scaling Principles**

LLMs adhere to several scaling principles that contribute to their impressive performance:

- **Model size**: Increasing the number of parameters in the model generally leads to improved performance across a wide range of tasks.

- **Dataset size**: Training on larger and more diverse datasets enhances the model's ability to generalize and understand context.
- **Compute resources**: The amount of computational power used during training significantly impacts the model's capabilities.

3.1.2 How LLMs Process and Generate Text

LLMs process and generate text through a series of sophisticated steps, leveraging their large-scale neural networks to understand and produce human-like language. Such process has been well-documented in recent Literature. [22]

1. **Input Processing**

When presented with input text, LLMs first tokenize the input into smaller units, typically subwords or characters. These tokens are then embedded into a high-dimensional vector space, where similar tokens are represented by vectors close to each other. (See Section 3.2 for more details on vector embeddings.)

2. **Contextual Understanding**

The embedded tokens are processed through multiple layers of the transformer architecture. Each layer refines the representation of the tokens, incorporating contextual information from the entire input sequence. Such process allows the model to capture long-range dependencies and nuanced relationships within the text.

3. **Text Generation**

Text generation in LLMs is typically performed using autoregressive methods:

1. The model predicts the probability distribution of the next token based on the input and previously generated tokens.
2. A token is selected from the distribution, often using techniques like nucleus sampling or temperature-controlled sampling to balance between diversity and coherence.
3. The selected token is appended to the output and fed back into the model as part of the input for the next prediction.
4. The process continues until a stopping condition is met, such as reaching a maximum length or generating a specific end token.

3.1.3 Theoretical Capabilities and Limitations

LLMs have demonstrated remarkable capabilities across various natural language processing tasks

- **Language understanding**: LLMs can comprehend complex linguistic structures, context, and nuances across multiple languages.
- **Task generalization**: These models can perform well on a wide range of tasks without task-specific fine-tuning, demonstrating strong few-shot and zero-shot learning abilities.
- **Knowledge integration**: LLMs can integrate and synthesize information from their training data, effectively serving as large-scale knowledge bases.
- **Creative generation**: LLMs can generate coherent and creative text across various styles and formats.

but LLMs also face inherent limitations [23]. These limitations are detailed below:

- **Lack of true understanding**: Despite their impressive performance, LLMs do not possess true comprehension or reasoning capabilities comparable to human cognition.
- **Hallucination**: These models can generate false or inconsistent information, especially when dealing with topics beyond their training data.
- **Contextual boundaries**: LLMs have limited ability to maintain context over very long sequences or across separate interactions.
- **Bias and fairness**: Models can perpetuate or amplify biases present in their training data, raising ethical concerns about their deployment.
- **Computational requirements**: The scale of these models necessitates significant computational resources for training and inference, limiting their accessibility.

3.2 Vector Embeddings

The following section dives into the theoretical foundations of vector embeddings, providing an overview of the mathematical concepts, dimensionality reduction techniques, and similarity measures used in vector spaces.

3.2.1 Mathematical Concepts

Vector embeddings have emerged as a powerful technique for representing textual data in machine learning and natural language processing. At their core, these embeddings map discrete linguistic units (such as words, phrases, or documents) to continuous vector spaces, enabling sophisticated mathematical operations and analyses.

The fundamental principle underlying vector embeddings is the distributional hypothesis, which posits that words appearing in similar contexts tend to have similar meanings. The hypothesis is operationalized by constructing dense vector representations that capture semantic and syntactic relationships between linguistic units.

Formally, let V be a vocabulary of size $|V|$. A vector embedding model aims to learn a function $f : V \to R^d$, where d is the dimensionality of the embedding space. For each word $w \in V$, the corresponding embedding vector $v_w = f(w)$ encodes its semantic and syntactic properties. Several mathematical frameworks have been developed to learn these embeddings:

1. **Matrix Factorization**: Methods like Latent Semantic Analysis (LSA) factorize word-context co-occurrence matrices to derive low-dimensional representations.
2. **Neural Network-based Approaches**: Models such as Word2Vec [24] and GloVe [25] utilize shallow neural networks to predict words from their contexts (or vice versa), learning embeddings as a byproduct of the prediction task.
3. **Contextual Embeddings**: Advanced models like BERT [26] employ deep bidirectional transformers to generate context-dependent representations, capturing nuanced word usage across different contexts.

The effectiveness of these embeddings stems from their ability to encode semantic relationships in the geometry of the vector space. For instance, analogical relationships often manifest as vector arithmetic operations (e.g., $v_{\text{king}} - v_{\text{man}} + v_{\text{woman}} \approx v_{\text{queen}}$).

3.2.2 Dimensionality Reduction Techniques

While the raw dimensionality of embedding spaces can be quite high, dimensionality reduction techniques are often employed to enhance

computational efficiency and mitigate the curse of dimensionality. These methods aim to preserve the most salient features of the high-dimensional embeddings while projecting them onto lower-dimensional subspaces.

Key dimensionality reduction techniques include:

1. **Principal Component Analysis (PCA)**: PCA identifies orthogonal axes (principal components) that capture the maximum variance in the data. Formally, given a set of n-dimensional vectors $\{x_1, ..., x_m\}$, PCA finds a transformation matrix W that projects these vectors onto a k-dimensional subspace $(k < n)$ while maximizing the variance of the projected data.
2. **t-Distributed Stochastic Neighbor Embedding (t-SNE)** [27]: t-SNE (non-linear) technique focuses on preserving local neighborhood structures in the high-dimensional space. t-SNE minimizes the Kullback-Leibler divergence between probability distributions representing pairwise similarities in the original and reduced spaces.
3. **Autoencoders** [28]: These neural network architectures learn compact representations by training to reconstruct input data through a bottleneck layer. The activations at the bottleneck serve as reduced-dimensional embeddings.
4. **Uniform Manifold Approximation and Projection (UMAP)** [29]: UMAP constructs a topological representation of the high-dimensional data and optimizes a low-dimensional layout that preserves the structure. UMAP often provides a favorable balance between global and local structure preservation.

The choice of dimensionality reduction technique depends on the specific requirements of the downstream task, such as visualization, clustering, or further machine learning applications.

3.2.3 Similarity Measures in Vector Spaces

A crucial aspect of working with vector embeddings is quantifying the similarity or distance between vectors. These similarity measures form the basis for numerous natural language processing tasks, including information retrieval, document classification, and semantic search. These measures are very well documented in literature [30].

Common similarity measures include:

1. **Cosine Similarity**: This metric measures the cosine of the angle between two vectors, providing a scale-invariant measure of orientation similarity. For vectors u and v, cosine similarity is defined as:

$$\text{cosine\_similarity}(A, B) = \frac{A \cdot B}{\|A\| \, \|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \sqrt{\sum_{i=1}^{n} (B_i)^2}} \tag{3.1}$$

Cosine similarity ranges from −1 (perfectly dissimilar) to 1 (perfectly similar), with 0 indicating orthogonality.

1. **Euclidean Distance**: This measure quantifies the straight-line distance between two points in the vector space. For n-dimensional vectors u and v, it is defined as:

$$\text{euclidean\_distance}(A, B) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2} \tag{3.2}$$

3. **Manhattan Distance**: Also known as L1 distance, this metric sums the absolute differences along each dimension:

$$\text{manhattan\_distance}(A, B) = \sum_{i=1}^{n} |A_i - B_i| \tag{3.3}$$

The choice of similarity measure can significantly impact the performance of downstream tasks and should be selected based on the specific properties of the embedding space and the requirements of the application.

3.3 Information Retrieval

Information Retrieval (IR) is a fundamental area of study in computer science that focuses on efficiently finding and presenting relevant information from large collections of data. This section explores key aspects of IR theory, including the comparison between classical and neural models, relevance ranking algorithms, and evaluation metrics for retrieval systems.

Information retrieval models can be broadly categorized into classical and neural approaches. Each category has its strengths and weaknesses, and understanding their differences is crucial for designing effective IR systems.

3.3.1 Classical IR Models

Classical IR models have been the foundation of information retrieval for decades. These models typically rely on statistical and probabilistic approaches to determine the relevance of documents to a given query.

1. **Boolean Model**: This model uses Boolean logic to match documents to queries. Documents are represented as sets of terms, and queries are formulated using Boolean operators (AND, OR, NOT). While simple and efficient, it lacks the ability to rank results.
2. **Vector Space Model (VSM)** [31]: In VSM, both documents and queries are represented as vectors in a high-dimensional space. Each dimension corresponds to a term in the vocabulary. Relevance is determined by calculating the cosine similarity between the query vector and document vectors. VSM allows for ranking of results and partial matching.
3. **Probabilistic Models**: These models, such as the Binary Independence Model (BIM) and BM25, estimate the probability of a document being relevant to a query. The mentioned models consider factors like term frequency, inverse document frequency, and document length to compute relevance scores.

3.3.2 Neural IR Models

Neural IR models leverage deep learning techniques to improve retrieval performance. These models can capture semantic relationships and contextual information more effectively than classical approaches.

1. **Embedding-based Models**: These models use neural networks to learn dense vector representations (embeddings) of words, sentences, or entire documents. Examples include Word2Vec [24], BERT [26], and Sentence-BERT. These embeddings capture semantic similarities, allowing for more nuanced matching between queries and documents.
2. **Neural Ranking Models**: These models directly learn to rank documents given a query. Such models can be categorized into:

- **Representation-based**: Learn separate representations for queries and documents, then compute relevance scores (e.g., DSSM, CDSSM).
- **Interaction-based**: Model the interactions between query and document terms explicitly (e.g., DRMM, K-NRM).
- **Hybrid**: Combine both representation and interaction-based approaches (e.g., DUET).

1. **End-to-End Neural IR Systems**: Recent advancements have led to the development of end-to-end neural IR systems that handle both candidate generation and ranking. Examples include ANCE and DPR, which use dense retrievers followed by neural re-rankers.

3.3.3 Relevance Ranking Algorithms

Relevance ranking is a critical component of IR systems, determining the order in which retrieved documents are presented to users. Several algorithms have been developed to address this challenge:

1. **Term Frequency-Inverse Document Frequency**

TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents [32]. It is the product of two components:

1. **Term Frequency (TF)**: Measures how frequently a term appears in a document.
2. **Inverse Document Frequency (IDF)**: Measures the importance of a term across the entire document collection.

This algorithm favors terms that are frequent in the current document but rare across the entire collection.

2. **BM25 (Best Matching 25)**

BM25 is a probabilistic ranking function that improves upon the basic TF-IDF model. It introduces document length normalization and saturation of term frequency. The BM25 score for a document D given a query Q is:

$$\frac{f(q_i, D)}{k_1 \times \left(1 - b + b \times \frac{|D|}{\text{avgdl}}\right) + f(q_i, D)} \tag{3.4}$$

$$\text{BM25}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \times (k_1 + 1) \times (3.4) \tag{3.5}$$

Where:

- $f(q_i, D)$ is the frequency of query term q_i in document D
- $|D|$ is the length of document D
- avgdl is the average document length in the collection
- k_1 and b are free parameters

BM25 is widely used in practice due to its effectiveness and efficiency.

3. **Learning to Rank (LTR)**

Learning to Rank is a machine learning approach to building ranking models for IR systems. It uses supervised learning techniques to train a model that can rank documents based on their relevance to a given query. LTR algorithms can be categorized into three main approaches:

1. **Pointwise Approach**: Treats document ranking as a regression or classification problem for single documents.
2. **Pairwise Approach**: Considers the relative order between pairs of documents and formulates ranking as a classification problem on document pairs.
3. **Listwise Approach**: Directly optimizes the order of an entire list of documents.

Popular LTR algorithms include RankNet, LambdaRank, and LambdaMART.

4. **Neural Ranking Models**

As mentioned earlier, neural ranking models have gained prominence in recent years. These models can be either representation-based or interaction-based:

1. **Representation-based Models**: Learn dense vector representations of queries and documents independently, then compute relevance scores using similarity measures (e.g., cosine similarity).
2. **Interaction-based Models**: Model the interactions between query and document terms explicitly, often using techniques like attention mechanisms or convolutional neural networks. [33]

Neural ranking models have shown strong performance in various IR tasks, particularly when large amounts of training data are available.

3.3.4 Evaluation Metrics

Evaluating the performance of IR systems is crucial for understanding their effectiveness and comparing different approaches [34]. The following are some common evaluation metrics used in IR:

1. **Precision and Recall**

- **Precision**: The fraction of retrieved documents that are relevant.

$$\text{Precision} = \frac{\text{Relevant Retrieved Documents}}{\text{Total Retrieved Documents}} \tag{3.6}$$

- **Recall**: The fraction of relevant documents that are retrieved.

$$\text{Recall} = \frac{\text{Relevant Retrieved Documents}}{\text{Total Relevant Documents}} \tag{3.7}$$

2. **F1 Score**

The F1 score is the harmonic mean of precision and recall:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.8}$$

1. **Mean Average Precision (MAP)**

MAP provides a single-figure measure of quality across recall levels. For a set of queries, MAP is the mean of the average precision scores for each query:

$$\text{MAP} = \frac{1}{Q} \sum_{q=1}^{Q} \text{AveP}(q) \tag{3.9}$$

Where $\text{AveP}(q)$ is the average precision for a single query.

4. **Normalized Discounted Cumulative Gain (nDCG)**

nDCG measures the usefulness, or gain, of a document based on its position in the result list. It uses a graded relevance scale and discounts the relevance of documents lower in the ranking:

$$\text{nDCG}_p = \frac{\text{DCG}_p}{\text{IDCG}_p} \tag{3.10}$$

Where:

- DCG_p is the Discounted Cumulative Gain at position p

- IDCG_p is the Ideal DCG at position p

5. **Mean Reciprocal Rank (MRR)**

MRR is the average of the reciprocal ranks of the first relevant document for a set of queries:

$$\text{MRR} = \left(\frac{1}{Q}\right) \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i} \tag{3.11}$$

Where rank_i is the rank position of the first relevant document for the i-th query.

3.4 Natural Language Processing

Natural Language Processing (NLP) is a multidisciplinary field at the intersection of computer science, artificial intelligence, and linguistics. It focuses on enabling computers to understand, interpret, and generate human language in a way that is both meaningful and useful. This section explores the foundational aspects of NLP, including the linguistic theories that underpin it, fundamental NLP tasks, and the processes of semantic and syntactic analysis.

3.4.1 Linguistic Theories

The development of NLP is deeply rooted in linguistic theories that provide frameworks for understanding the structure and meaning of language. Some key linguistic theories that have significantly influenced NLP include:

- **Chomsky's Generative Grammar**: This theory posits that language is governed by a set of structural rules that can generate all possible grammatical sentences in a language. It has been instrumental in developing formal grammars for natural language parsing.
- **Cognitive Linguistics**: This approach emphasizes the relationship between language, mind, and socio-physical experience. It has influenced the development of semantic models and frame-based knowledge representation in NLP.
- **Distributional Semantics**: This theory suggests that words that occur in similar contexts tend to have similar meanings. It forms the basis for many modern word embedding techniques, such as Word2Vec [24] and GloVe [25].

- **Discourse Analysis**: This field studies language use in context, considering how larger units of language create meaning. It has applications in dialogue systems and text coherence analysis.

These theories provide the conceptual foundation for many NLP algorithms and models, guiding the development of computational approaches to language processing.

3.4.2 Fundamental NLP Tasks

NLP encompasses a wide range of tasks, each addressing different aspects of language processing. Some of the most fundamental tasks include:

- **Tokenization**: The process of breaking down text into individual units (tokens), typically words or subwords. This is often the first step in many NLP pipelines.
- **Part-of-Speech (POS) Tagging**: Assigning grammatical categories (e.g., noun, verb, adjective) to each word in a text. This task is crucial for understanding the syntactic structure of sentences.
- **Parsing**: Analyzing the grammatical structure of sentences. This can involve constituency parsing (breaking sentences into nested constituents) or dependency parsing (identifying grammatical relationships between words).
- **Named Entity Recognition (NER)**: Identifying and classifying named entities (e.g., person names, organizations, locations) in text. This is essential for information extraction and question answering systems.
- **Coreference Resolution**: Determining when different expressions in a text refer to the same entity. This is crucial for maintaining coherence in language understanding.
- **Sentiment Analysis**: Determining the sentiment or emotional tone of a piece of text, often categorized as positive, negative, or neutral.
- **Machine Translation**: Automatically translating text from one language to another, a task that integrates many aspects of NLP.

These tasks form the building blocks for more complex NLP applications, such as chatbots, text summarization systems, and question-answering systems.

3.4.3 Semantic and Syntactic Analysis

Semantic and syntactic analyses are core processes in NLP that deal with extracting meaning and structure from text:

Syntactic Analysis: Syntactic analysis focuses on the grammatical structure of sentences. It involves:

- **Constituency Parsing**: Breaking down sentences into nested constituents (e.g., noun phrases, verb phrases) to form a parse tree.
- **Dependency Parsing**: Identifying grammatical relationships between words in a sentence, often represented as a directed graph.
- **Shallow Parsing**: Identifying the main syntactic components of a sentence without specifying their internal structure or relations.

Syntactic analysis is crucial for tasks that require understanding sentence structure, such as grammar checking and machine translation.

Semantic Analysis: Semantic analysis aims to extract meaning from text. It encompasses:

- **Word Sense Disambiguation**: Determining the correct meaning of a word in a given context.
- **Semantic Role Labeling**: Identifying the semantic roles of words in a sentence (e.g., agent, patient, instrument).
- **Entailment and Contradiction Detection**: Determining the logical relationship between sentences or propositions.
- **Semantic Parsing**: Mapping natural language to a formal meaning representation, such as logical forms or database queries.

Semantic analysis is essential for applications that require deep understanding of text, such as question answering systems and text summarization.

The interplay between syntactic and semantic analysis is crucial in NLP. While syntactic analysis provides the structural framework, semantic analysis fills in the meaning, allowing for a comprehensive understanding of natural language.

Advanced NLP models, particularly those based on deep learning, often learn to perform both types of analysis implicitly through training on large datasets.

3.5 Database Theory

The following section provides an overview of the database theory concepts that underpin the design and implementation of AIKO.

3.5.1 Relational Database Concepts

Relational databases have been the cornerstone of data management systems for decades [35], providing a structured approach to organizing and querying data. The relational model, introduced by E.F. Codd in 1970, is based on the mathematical concept of relations and set theory.

The relational model is built upon several fundamental principles:

1. **Data Organization**: Data is organized into tables (relations) consisting of rows (tuples) and columns (attributes).
2. **Data Integrity**: Constraints such as primary keys, foreign keys, and unique constraints ensure data consistency and accuracy.
3. **Data Independence**: Logical and physical data independence allows for changes in the database schema or storage mechanisms without affecting application logic.
4. **Declarative Query Language**: SQL (Structured Query Language) provides a high-level, declarative means of data manipulation and retrieval.

- **Normalization**

Normalization is a critical process in relational database design, aimed at minimizing data redundancy and improving data integrity. The most commonly used normal forms are:

- First Normal Form (1NF): Eliminate repeating groups and ensure atomic values.
- Second Normal Form (2NF): Remove partial dependencies on the primary key.
- Third Normal Form (3NF): Eliminate transitive dependencies.

Higher normal forms, such as Boyce-Codd Normal Form (BCNF) and Fourth Normal Form (4NF), address more specific anomalies but are less commonly used in practice.

- **Relational Algebra**

Relational algebra forms the theoretical foundation for operations in relational databases. Key operations include:

- Selection (σ): Filters rows based on a condition.
- Projection (π): Selects specific columns from a relation.
- Union (∪): Combines tuples from two relations.
- Intersection (∩): Retains only tuples common to both relations.
- Difference (-): Removes tuples from one relation that appear in another.
- Cartesian Product (×): Combines every tuple from one relation with every tuple from another.
- Join (⋈): Combines related tuples from two relations based on a join condition.

These operations provide the basis for more complex queries and data manipulations in relational database systems.

3.5.2 NoSQL and Vector Databases

As data volumes and variety have grown exponentially, traditional relational databases have faced scalability and flexibility challenges. NoSQL (Not Only SQL) databases emerged as a response to these challenges, offering alternative data models and relaxed consistency guarantees to achieve better performance and scalability. Vector databases, a specialized type of database, are designed to store and query high-dimensional vector data efficiently, making them ideal for machine learning and similarity search applications.

- **NoSQL Data Models**

NoSQL databases typically fall into four main categories:

1. **Key-Value Stores**: Simple databases that store data as key-value pairs (e.g., Redis, DynamoDB).
2. **Document Stores**: Store and query data in document-like structures, often using JSON (e.g., MongoDB, CouchDB).
3. **Column-Family Stores**: Organize data into column families, optimized for queries over large datasets (e.g., Cassandra, HBase).

4. **Graph Databases**: Represent and store data as nodes and edges, optimized for highly connected data (e.g., Neo4j, JanusGraph).

- **CAP Theorem**

The CAP theorem, introduced by Eric Brewer, states that it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:

- Consistency: All nodes see the same data at the same time.
- Availability: Every request receives a response, without guarantee that it contains the most recent version of the information.
- Partition Tolerance: The system continues to operate despite arbitrary partitioning due to network failures.

NoSQL systems often prioritize availability and partition tolerance over strict consistency, leading to eventual consistency models.

- **Vector Databases**

Vector databases are a specialized type of database designed to store and query high-dimensional vector data efficiently. Vector Databases are particularly useful in machine learning and artificial intelligence applications, where data is often represented as dense vectors.

Key features of vector databases include:

1. **Efficient Similarity Search**: Ability to perform fast nearest neighbor searches in high-dimensional spaces.
2. **Indexing Techniques**: Use of specialized indexing methods like HNSW (Hierarchical Navigable Small World) or IVF (Inverted File) to speed up similarity queries.
3. **Scalability**: Designed to handle large volumes of high-dimensional data efficiently.

Examples of vector databases include Pinecone [36], Milvus, and Faiss (while not a full database, it provides similar functionality for vector search).

3.5.3 ACID Properties and Eventual Consistency

ACID properties are a set of guarantees that ensure reliable processing of database transactions. ACID properties are particularly important in systems where data integrity is crucial, such as financial applications.

1. **Atomicity**: A transaction is treated as a single, indivisible unit that either completes entirely or fails completely.
2. **Consistency**: A transaction brings the database from one valid state to another, maintaining all predefined rules and constraints.
3. **Isolation**: Concurrent execution of transactions results in a state that would be obtained if transactions were executed sequentially.
4. **Durability**: Once a transaction is committed, it will remain so, even in the event of power loss, crashes, or errors.

Most relational database management systems (RDBMS) provide strong ACID guarantees, ensuring data integrity at the cost of potential performance limitations in distributed environments. In contrast, NoSQL databases often relax some ACID properties to achieve better scalability and availability. The tradeoff between strong consistency and performance is a key consideration in database design.

- **Eventual Consistency**

Eventual consistency is a consistency model used in distributed systems that allows for temporary inconsistencies but guarantees that all replicas will eventually converge to a consistent state, given no further updates.

Key aspects of eventual consistency include:

1. **Conflict Resolution**: Mechanisms to resolve conflicts when multiple updates occur simultaneously on different replicas.
2. **Anti-Entropy Protocols**: Background processes that synchronize data across replicas to ensure convergence.
3. **Vector Clocks**: A technique used to track the causal relationships between different versions of data across distributed replicas.

Eventual consistency is often employed in NoSQL systems to achieve better scalability and availability, particularly in scenarios where absolute consistency is not critical or where the time window for inconsistency is acceptably small.

Between the strong consistency provided by ACID transactions and the weak guarantees of eventual consistency, there exists a spectrum of consistency models:

1. **Strong Consistency**: All replicas return the same value for a read operation at a given time.
2. **Sequential Consistency**: All operations appear to occur in some sequential order, consistent with the order seen by individual processes.
3. **Causal Consistency**: Operations that are causally related are seen by every node in the same order.
4. **Eventual Consistency**: After a period of no updates, all replicas will gradually become consistent.

The choice of consistency model depends on the specific requirements of the application, balancing factors such as data integrity, availability, and performance.

3.6 Web Application Architecture

Web application architecture has evolved significantly since the inception of the World Wide Web. This section examines three fundamental aspects of modern web application architecture: the client-server model, RESTful architecture principles, and the spectrum of rendering approaches from server-side to client-side, including hybrid solutions.

3.6.1 Client-Server Model

The client-server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. This model underpins the vast majority of web applications and has several key characteristics.

The two components of the client-server model are:

- **Client**: Typically a web browser or mobile application that sends requests to the server and displays the response to the user.

- **Server**: A computer or system that hosts the web application, processes requests from clients, and sends back responses.

Communication between client and server occurs over a computer network, most commonly the Internet, using standardized protocols such as HTTP or HTTPS.

They have several advantages

- Clear separation of concerns
- Centralized data storage and management
- Scalability through load distribution
- Easier maintenance and updates

and associated challenges.

- Network latency
- Server availability and reliability
- Potential single point of failure

3.6.2 RESTful Architecture Principles

Representational State Transfer (REST) is an architectural style for distributed hypermedia systems, introduced by Roy Fielding in his doctoral dissertation (2000). RESTful architecture has become the de facto standard for web APIs due to its simplicity, scalability, and performance.

- **Key Principles**

1. **Statelessness**: Each request from client to server must contain all information necessary to understand and process the request.
2. **Client-Server**: A uniform interface separates clients from servers, allowing independent evolution of application functions.
3. **Cacheable**: Responses must implicitly or explicitly define themselves as cacheable or non-cacheable to prevent clients from reusing stale or inappropriate data.
4. **Uniform Interface**: A constraint defined by four interface constraints:
   - Identification of resources
   - Manipulation of resources through representations
   - Self-descriptive messages

- Hypermedia as the engine of application state (HATEOAS)

5. **Layered System**: A client cannot ordinarily tell whether it is connected directly to the end server or an intermediary along the way.

RESTful architecture offers several benefits:

- Improved scalability due to stateless operations
- Enhanced visibility and reliability through the uniform interface
- Independent evolution of client and server components
- Reduced coupling between client and server

3.6.3 Server-side vs. Client-side vs. Hybrid Rendering

The choice of rendering approach significantly impacts a web application's performance, user experience, and development complexity.

3.6.3.1 Server-side Rendering (SSR)

In SSR, the server processes requests, generates HTML, and sends the fully rendered page to the client.

- **Advantages:**
  - Faster initial page load
  - Better SEO as content is immediately available to search engines
  - Reduced client-side processing requirements
- **Disadvantages:**
  - Higher server load
  - Slower subsequent page loads due to full page reloads
  - Less interactive user experience
- **Client-side Rendering (CSR)**

CSR relies on JavaScript running in the browser to render content dynamically.

- **Advantages:**
  - Rich, interactive user experiences
  - Reduced server load
  - Faster subsequent page loads

- **Disadvantages:**
  - Slower initial page load
  - Potential SEO challenges
  - Higher client-side resource requirements
- **Hybrid Approaches**

  Hybrid approaches aim to combine the benefits of both SSR and CSR.
- **Hydration**

  Hydration involves sending a pre-rendered HTML page from the server, along with the JavaScript necessary to make it interactive. The client-side JavaScript then "hydrates" the static HTML, attaching event listeners and making the page fully interactive.
- **Advantages:**
  - Fast initial page load with SSR
  - Smooth transition to a fully interactive CSR application
- **Disadvantages:**
  - Potential for layout shifts during hydration
  - Increased complexity in development and debugging
- **Resumability**

  Resumability is an emerging technique that aims to improve upon hydration. Instead of re-executing all JavaScript on the client, a resumable application can "resume" from where the server left off.
- **Advantages:**
  - Eliminates redundant work between server and client
  - Potentially faster and more efficient than traditional hydration
- **Disadvantages:**
  - Relatively new concept with limited widespread adoption
  - Increased complexity in state management between server and client

3.7 Security and Authentication

In the realm of modern computing and networked systems, security and authentication play pivotal roles in safeguarding digital assets, maintaining data integrity, and

ensuring authorized access. This section delves into three crucial aspects of security and authentication: cryptographic principles, the OAuth 2.0 framework, and the zero trust security model.

3.7.1 Cryptographic Principles

Cryptography forms the bedrock of information security, providing mechanisms to protect data confidentiality, integrity, and authenticity. At its core, cryptography involves the use of mathematical algorithms to transform plaintext into ciphertext, rendering it unintelligible to unauthorized parties.

- Symmetric and Asymmetric Encryption

Symmetric encryption, also known as secret-key cryptography, utilizes a single key for both encryption and decryption processes. While efficient for large-scale data encryption, it faces challenges in secure key distribution. Notable symmetric algorithms include AES (Advanced Encryption Standard) and ChaCha20.

Asymmetric encryption, or public-key cryptography, employs a pair of mathematically related keys: a public key for encryption and a private key for decryption. This approach resolves the key distribution problem but is computationally intensive. RSA (Rivest-Shamir-Adleman) and elliptic curve cryptography (ECC) are prominent examples of asymmetric algorithms.

- Hash Functions and Digital Signatures

Cryptographic hash functions play a crucial role in ensuring data integrity. These one-way functions map input data of arbitrary length to fixed-size output values, known as hash digests. Ideal hash functions exhibit properties such as collision resistance and the avalanche effect. SHA-256 and Blake2 are widely adopted hash functions.

Digital signatures, built upon asymmetric cryptography and hash functions, provide a means of verifying message authenticity and non-repudiation. The process involves encrypting a message digest with the sender's private key, allowing recipients to verify the signature using the corresponding public key.

3.7.2 OAuth 2.0 Framework

The OAuth 2.0 (Open Authorization) framework [20] has emerged as a de facto standard for secure delegation of access to protected resources in distributed systems. It enables third-party applications to obtain limited access to user accounts on HTTP services, without necessitating the sharing of long-term credentials.

- **Core Concepts and Roles**

OAuth 2.0 defines four primary roles:

1. **Resource Owner**: The entity capable of granting access to a protected resource.
2. **Resource Server**: The server hosting the protected resources.
3. **Client**: The application requesting access to protected resources.
4. **Authorization Server**: The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

- **Authorization Grant Types**

The framework specifies multiple authorization types to accommodate various cases:

1. **Authorization Code Grant**: Optimized for confidential clients, this flow involves user authentication and consent, followed by the exchange of an authorization code for an access token.
2. **Implicit Grant**: Designed for public clients operating in web browsers, this flow directly issues access tokens to the client.
3. **Resource Owner Password Credentials Grant**: Allows direct exchange of user credentials for an access token, suitable for trusted first-party applications.
4. **Client Credentials Grant**: Enables client authentication and authorization based on its own credentials, typically used for machine-to-machine communication.

- **Security Considerations**

While OAuth 2.0 provides a robust framework for authorization, its security relies heavily on proper implementation. Key considerations include:

- Secure communication channels (TLS)
- Protection against cross-site request forgery (CSRF) attacks

- Proper validation and storage of tokens
- Implementation of token expiration and revocation mechanisms

3.7.3 Zero Trust Security Model

The zero trust security model represents a paradigm shift in cyber-security, moving away from traditional perimeter-based security approaches. This model operates on the principle of "never trust, always verify," assuming that threats may exist both outside and inside the network perimeter.

- **Core Principles**

1. Verify explicitly: Authenticate and authorize based on all available data points, including user identity, device health, and application state.
2. Use least privilege access: Limit user access with just-in-time and just-enough-access (JIT/JEA), risk-based adaptive policies.
3. Assume breach: Minimize blast radius for breaches and prevent lateral movement by segmenting access by network, user, devices, and application awareness.

- **Implementation Strategies**

Implementing a zero trust architecture involves several key strategies:

1. **Strong identity verification**: Employ multi-factor authentication (MFA) and continuous authentication mechanisms.
2. **Device health validation**: Assess device security posture before granting access to resources.
3. **Micro-segmentation**: Divide the network into small zones to maintain separate access for separate parts of the network.
4. **Least privilege access**: Grant users only the access, explicitly needed to perform their tasks.
5. **Data-centric security**: Focus on protecting data, both at rest and in transit, rather than just securing network segments.

- **Challenges and Considerations**

The zero trust model presents with the following challenges:

- **Legacy system integration**: Adapting existing infrastructure to support zero trust principles can be complex.
- **Performance impact**: Continuous authentication and authorization checks may introduce latency.
- **User experience**: Balancing security with usability requires careful design of authentication workflows.

3.8 Software Engineering Principles

The Software Engineering Principles underlined below, were used throughout the development of the project to ensure the quality, maintainability, and scalability of the codebase. Readers should keep these principles in mind when reviewing the project's codebase.

3.8.1 Design Patterns

Design patterns are reusable solutions to common problems in software design. Design patterns provide a standardized approach to solving specific issues, enhancing code readability, maintainability, and scalability. The current section focuses on design patterns relevant to various architectural approaches, specifically the Adapter, Bridge, Decorator, Facade, Template Method, and Iterator patterns.

- **Adapter Pattern**

The Adapter pattern allows incompatible interfaces to work together. It acts as a bridge between two incompatible interfaces by converting the interface of a class into another interface that clients expect. This pattern is particularly useful in systems with legacy code or when integrating new components into existing systems.

- **Bridge Pattern**

The Bridge pattern decouples an abstraction from its implementation, allowing both to vary independently. This pattern is especially useful when both the abstraction and its implementation need to be extended using subclasses. It promotes loose coupling and enhances the overall flexibility of the system.

- **Decorator Pattern**

The Decorator pattern allows behavior to be added to individual objects dynamically without affecting the behavior of other objects from the same class. This pattern is used to extend or alter the functionality of objects at runtime, providing a flexible alternative to subclassing for extending functionality.

- **Facade Pattern**

The Facade pattern provides a unified interface to a set of interfaces in a subsystem. It defines a higher-level interface that makes the subsystem easier to use by reducing complexity and minimizing the communication and dependencies between subsystems. This pattern is particularly useful in complex systems where subsystems are tightly coupled.

- **Template Method Pattern**

The Template Method pattern defines the skeleton of an algorithm in a method, deferring some steps to subclasses. It allows subclasses to redefine certain steps of an algorithm without changing the algorithm's structure. This pattern is commonly used in frameworks where the overall structure of an algorithm is fixed, but specific steps can be customized.

- **Iterator Pattern**

The Iterator pattern provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation. This pattern is widely used in collection frameworks and allows for traversing different collections uniformly without exposing their internal structure.

3.8.2 SOLID Principles

SOLID is an acronym for five design principles intended to make software designs more understandable, flexible, and maintainable. These principles are fundamental to object-oriented programming and design.

- **Single Responsibility Principle (SRP)**

The SRP states that a class should have only one reason to change, meaning it should have only one job or responsibility. This prin-

ciple promotes high cohesion and helps in managing complexity by ensuring that each class focuses on a specific aspect of the system.

- **Open-Closed Principle (OCP)**

The OCP suggests that software entities (classes, modules, functions, etc.) should be open for extension but closed for modification. This principle encourages the use of abstractions and polymorphism to allow new functionality to be added with minimal changes to existing code.

- **Liskov Substitution Principle (LSP)**

The LSP states that objects of a superclass should be replaceable with objects of its subclasses without affecting the correctness of the program. This principle ensures that inheritance is used correctly and promotes the creation of well-structured class hierarchies.

- **Interface Segregation Principle (ISP)**

The ISP advises that no client should be forced to depend on methods it does not use. This principle suggests breaking down large interfaces into smaller, more specific ones, allowing clients to only know about the methods that are of interest to them.

- **Dependency Inversion Principle (DIP)**

The DIP states that high-level modules should not depend on low-level modules; both should depend on abstractions. Additionally, abstractions should not depend on details; details should depend on abstractions. This principle promotes loose coupling and facilitates easier testing and maintenance of code.

3.8.3 Microservices Architecture Theory

Microservices architecture is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often HTTP/REST APIs. This architectural style has gained significant popularity due to its ability to support large, complex applications while maintaining scalability and flexibility.

- **Key Characteristics**

1. **Decomposition by Business Capability**: Services are organized around business capabilities, promoting a clear separation of concerns.
2. **Autonomy**: Each service is developed, deployed, and scaled independently, allowing for greater flexibility and faster development cycles.
3. **Decentralized Data Management**: Each service manages its own database, either different instances of the same database technology or entirely different database systems.
4. **Smart Endpoints and Dumb Pipes**: Microservices receive requests, process them, and produce a response, with simple protocols (often REST over HTTP) used for communication.
5. **Failure Isolation**: The failure of a single service does not cascade to bring down the entire application.

• **Benefits and Challenges**

Microservices architecture offers several benefits, including improved scalability, flexibility in technology stack selection, and easier maintenance. However, it also introduces challenges such as increased complexity in deployment and monitoring, potential performance overhead due to network communication, and the need for careful service boundary definition.

• **Implementation Considerations**

Implementation of microservices architecture requires careful consideriation:

1. **Service Discovery**: Mechanisms for services to locate and communicate with each other dynamically.
2. **API Gateway**: A single entry point for all clients, handling authentication, load balancing, and routing requests to appropriate services.
3. **Distributed Tracing**: Tools and practices to monitor and debug requests as the requests propagate through multiple services.
4. **Containerization and Orchestration**: Technologies like Docker and Kubernetes are often used to package and manage microservices.

5. **Event-Driven Architecture**: Many microservices implementations use event-driven models to handle asynchronous communication between services.

CHAPTER 4
METHODOLOGY

This section goes into detail about the overall system design, the development approach, the various “ends” of the system, and the technologies used. It also provides a short overview of the libraries and frameworks used in the development of AIKO.

4.1 High-Level Overview

AIKO comprises the following key subsystems and components:

4.1.1 Data Ingestion and Integration Subsystem

The Data Ingestion and Integration Subsystem serves as the primary mechanism for acquiring and harmonizing data from diverse sources into a unified format. The subsystem leverages specialized data connectors, application programming interfaces (APIs), and web scraping techniques to facilitate comprehensive information retrieval. Through sophisticated data transformation and normalization processes, the subsystem ensures seamless integration of all incoming data streams.

4.1.2 Information Processing Subsystem

The Information Processing Subsystem manages the complex task of processing and analyzing ingested data across multiple modalities. The subsystem employs advanced Natural Language Processing (NLP), Computer Vision capabilities—including Face Recognition and Object Detection—and Audio Processing algorithms. Through these technologies, the subsystem generates detailed metadata, tags, and annotations, enabling enhanced search and retrieval functionality.

4.1.3 Knowledge Base and Indexing Subsystem

The Knowledge Base and Indexing Subsystem maintains responsibility for the storage and indexing of processed data, ensuring efficient retrieval operations. The subsystem incorporates Vector Databases, Query Processing Engines, and sophisticated Indexing Algorithms, complemented by robust, secure, and scalable storage solutions. Critical to the subsystem’s operation are the implemented security measures and access control mechanisms, which provide comprehensive data protection.

4.1.4 Search and Retrieval Engine

The Search and Retrieval Engine delivers advanced search capabilities and personalized recommendations to end users. The engine incorporates Semantic Search functionality, NLP Query Understanding, and Data Ranking Algorithms to optimize search results. Users benefit from implemented faceted search capabilities, comprehensive filtering options, and customizable result presentation features.

4.1.5 User and Data Management Subsystem

The User and Data Management Subsystem orchestrates user profiles, access controls, and cross-device synchronization. The subsystem implements OAuth protocols and social login mechanisms to ensure secure authentication. Real-time synchronization capabilities and conflict resolution strategies maintain data consistency across platforms. Additionally, an integrated notification system manages user alerts and updates effectively.

4.1.6 User Interface and Experience Subsystem

The User Interface and Experience Subsystem delivers an intuitive, responsive, and user-friendly interface for system interaction. The subsystem adheres to modern web technologies, responsive design principles, and accessibility standards. The architecture (currently) encompasses two primary components: a Web Application and an API with comprehensive documentation and example code for third-party integrations and power users. Together, these components ensure a cohesive and accessible user experience across all interaction points.

4.2 System Interactions

The subsystems within AIKO interact with each other to provide a seamless user experience and efficient information management. The data ingestion and integration subsystem feeds data into the information processing subsystems, which generate metadata and annotations for storage in the knowledge base. The search and retrieval engine retrieves relevant information based on user queries and preferences, while the user management subsystem ensures secure access and synchronization

across devices. The user interface subsystem provides an intuitive interface for users to interact with the system and access the information stored in AIKO.

Figure 4.1: System Interactions

4.3 Development Approach

AIKO was developed using the Waterfall Methodology [37]. The project was divided into distinct phases, each culminating in a set of deliverables that were reviewed and approved before proceeding to the next phase. The Waterfall Methodology was chosen for its structured approach, clear milestones, and well-defined requirements. The development process followed the following stages:

4.3.1 Requirements Gathering

The project began with a comprehensive requirements gathering phase, where the team worked closely with stakeholders to define the scope, objectives, and key features of AIKO. The requirements were documented in detail to ensure a clear understanding of the project goals. A requirements specification document was created to serve as a reference throughout the development process [38]. The Software Requirements Specification (SRS) document outlined the functional and non-functional requirements of AIKO. A digital copy of the SRS document is available alongside the project source code.

The SRS detailed the requirements of the system as well as the methods of verification and validation required to ensure that the system met the specified requirements. It was approved on July 31, 2024, and served as the foundation for the design phase of AIKO.

4.3.2 Design and Architecture

The design phase of AIKO focused on translating the requirements outlined in the SRS document into a detailed system architecture and design. The team conducted a series of design workshops to define the system components, interactions, and data flows. The architecture was designed to be modular, scalable, and extensible, allowing for future enhancements and integrations. The design phase culminated in the creation of detailed design documents, including system diagrams, data models, and interface mockups. The Software Design Document (SDD) was created to document the architectural decisions, design patterns, and technologies used in AIKO [39]. It was approved on August 16, 2024, and served as a blueprint for the development phase.

The SDD detailed the high-level architecture of AIKO, including the subsystems, components, and interactions within the system. The set of technologies was (and is) quickly evolving and as such was not included in the SDD.

4.3.3 Development Phase

The development phase of AIKO involved the implementation of the system's components, features and functionalities. The team followed a semi-

structured approach. The development process was iterative. A simple version of the system was developed first, followed by incremental enhancements and additions. The details about the libraries, frameworks etc. are included in Section 4.5 and Section 4.7. The increments are detailed below:

- **Increment 1** : Setup the core project structure to be a monorepo [40] with the following packages:
  - `src-web`: The main frontend web application
  - `src-backend`: The main backend server
  - `supabase`: The package detailing the structure and setup of the Supabase infrastructure.
  - `support`: This package contains the support files and additional non-essentials of the project.
- **Increment 2**
  - Implementation of a novel, extensible communication stack for Large Language Models (LLMs).
  - Implementation of a plugin system for the Data Ingestion Service.
  - Addition of support for a few basic file formats such as `text/*` and `application/pdf`.
- **Increment 3**
  - Implementation of a simple user interface for authentication, user management and ingestion.
- **Increment 4**
  - Implementation of a simple search engine with basic search capabilities.
  - Implementation of a simple metadata generation system.
  - Implementation of a simple data storage system.
- **Increment 5**
  - Refinement of the search engine with advanced search capabilities.
  - Refinement of the metadata generation system to handle larger contexts (up to 32,768 tokens)

- Expansion of the plugin system to include the LLMs as well.

- **Increment 6**
  - Addition of support for more file formats such as `image/*` and `audio/*`.
  - Addition of a landing page for the web application.
  - Development of a notification library for the frontend.
- **Increment 7**
  - Extension of the DIS to include proper support for `.docx` and `.pptx` series of files.
  - Extension of the IPS to include proper support for `.docx` and `.pptx` series of files.
  - Addition of more model providers
- **Increment 8**
  - Implementation of a new model for the DIS and IPS to increase the context window to 128,000 tokens.
  - Fixed regression errors with the DIS and IPS's processing of edge case files.
- **Increment 9**
  - Implementation of a "visual-pdf" plugin, which allows for the system to consider graphics embedded within documents.
  - Addition of support for external sources in the DIS.
- **Increment 10**
  - Addition of an Chrome and Firefox Extension to save YouTube Video and Web Pages.
  - Integration of web extension with the DIS.

Unit and End-to-End tests were written and executed at the end of each increment to ensure the quality and reliability of the system. The development phase had to be completed ahead of schedule due to a change in the project timeline. The development phase was completed on September 12, 2024. It was scheduled to end on October 7, 2024. The early completion of the development phase required the team to defer some features.

4.3.4 Testing and Quality Assurance

The testing phase of AIKO began on September 12, 2024. The testing phase involved a series of unit tests, integration tests, and end-to-end tests to validate the

functionality and performance of the system. The testing procedures were outlined in the SRS [38] and SDD [39]. A comprehensive document detailing the testing procedures and results was created to ensure that the system met the specified requirements and functioned according to the design specifications. Issues with the system were documented and resolved promptly to ensure the quality and reliability of AIKO.

The Verification and Validation Document [41] was approved on September 19, 2024, signifying the completion of the testing phase.

4.3.5 Deployment and Maintenance

The final phase of AIKO required the deployment of the system to a production environment. The deployment process involved setting up the necessary infrastructure, configuring the servers, and ensuring that the system was operational and accessible to users. The deployment phase was completed on September 26, 2024, and AIKO was made available to a select user group for testing and feedback.

4.3.6 Project Budget

Another document [42] was created to detail the financial requirements of the project. The budget estimation included costs for development, testing, deployment, and maintenance of AIKO. The total budget for the project was estimated at ₹22,300 with about ₹21,000 allocated for the machine learning models and the rest for the hosting of the platform. The budget was approved on August 07, 2024. The following tables provide a brief overview of the project budget.

Table 4.1: Monthly Operating Costs

Item	**Cost per Month (INR)**
Machine Learning Cloud	₹5,000
API Access for GenAI models	₹3,000
Hosting - Backend	₹2,000
Total Monthly Costs	**₹10,000**

Table 4.2: One-time Costs

Item	Cost (INR)
Domain Purchase	₹1,300
Total One-time Costs	**₹1,300**

Table 4.3: Total Project Budget

Item	Cost (INR)
Monthly Operating Costs (₹10,000 × 8 months)	₹80,000
One-time Domain Purchase	₹1,300
Total Operating Cost	**₹81,300**

The following sections provide detailed information on the technologies and frameworks used in the development of AIKO.

4.4 Data Layer Implementation

The realtional database was hosted using Supabase [43]. Supabase is an open-source platform that provides a powerful alternative to Firebase. It offers a suite of tools and services to help developers build scalable and secure web and mobile applications quickly. Key features of Supabase include:

- Real-time database
- Authentication and user management
- Auto-generated APIs from database schemas.
- Serverless functions
- Storage for large files

Supabase is built on top of PostgreSQL [44], giving developers the flexibility of a robust relational database while providing the ease of use typically associated with Backend-as-a-Service (BaaS) platforms.

The exact database schema is given in Appendix B

4.4.1 Users

This table is used to store the user's metadata.

- id (UUID, Primary Key, References auth.users(id)): Unique Identifier
- name (TEXT) : User's Name
- theme (TEXT, Default: 'light'): User's Theme
- other (JSONB): Other Metadata
- updated_at (TIMESTAMP WITH TIME ZONE, Default: CURRENT_TIMESTAMP): Last Updated Time

4.4.2 Entities

This is the primary table in use by AIKO. It stores the entities that are ingested into the system.

- id (UUID, Primary Key, Default: uuid_generate_v4()) : Unique Identifier
- user (UUID, References auth.users(id)): User Identifier
- source (TEXT, Not Null): Source of the Entity
- type (TEXT, Not Null): Type of the Entity
- title (TEXT): Title of the Entity as generated by IPS
- description (TEXT): Description of the Entity as generated by IPS
- tags (TEXT[], Default: '{}'): Tags for the Entity as generated by IPS
- processed (BOOLEAN, Default: FALSE): Whether the Entity has been processed by the DIS
- processed_at (TIMESTAMP WITH TIME ZONE): Time of Processing by the DIS
- created_at (TIMESTAMP WITH TIME ZONE, Default: CURRENT_TIMESTAMP): Time of Creation
- updated_at (TIMESTAMP WITH TIME ZONE, Default: CURRENT_TIMESTAMP): Last Updated Time
- deleted (BOOLEAN, Default: FALSE): Whether the Entity has been deleted (used in case of soft deletion)
- metadata (JSONB): Additional Metadata, such as the model used for processing

4.4.3 Messages

This table stores the chat messages as generated by the user as well as the system.

- id (UUID, Primary Key, Default: uuid_generate_v4()): Unique Identifier
- user (UUID, References auth.users(id)): User Identifier
- entity (UUID, References public.entity(id)): Entity Identifier
- content (TEXT): Content of the Message
- is_user_message (BOOLEAN, Default: TRUE): Whether the Message is from the User
- created_at (TIMESTAMP WITH TIME ZONE, Default: CURRENT_TIMESTAMP): Time of Creation
- metadata (JSONB): Additional Metadata, such as the model used for processing

4.5 Backend Development

The backend of AIKO encapsulates the API Layer alongside the DIS and IPS as detailed in Section 4.1. The backend is responsible for handling data ingestion, processing, storage, and retrieval. This section delves into the technologies, frameworks, and design considerations that underpin the backend development of AIKO.

The backend consists of one main server that handles all incoming requests and manages the routing for the data that is ingested and processed. The backend server is authenticated with Supabase's User JWT and uses that to verify and route the various requests to the appropriate services. It also keeps track of the various models and plugins that are available for use in the DIS and IPS.

The backend is stateless, meaning that it does not store any session data. This allows for easy scaling and deployment across cloud environments. The backend server is designed to be highly scalable and fault-tolerant, ensuring that it can handle large volumes of data and requests efficiently.

The team initially wanted to use ready-to-use solutions for the backend, but due to a underlying need for customizability, performance and the stateless nature of the backend, the team decided to build the LLM communication stack, the plugin architecture and the API flows from scratch.

4.5.1 Novel LLM Communication Stack

The most used libraries for LLMs is LangChain [45]. The LangChain library is a simple, yet powerful library that allows for easy communication with LLMs. The library is designed to be extensible and flexible, allowing for easy integration with various LLMs. However, the abstractions over the LLMs are very high level, such that the extensible nature of AIKO would not have been possible if LangChain were used. The abstractions also cause a performance hit, which is not acceptable for AIKO. The team decided to build a custom communication stack using WebSockets and a custom protocol over WebSockets that would allow for easy integration with various LLMs, while also providing a low-level interface for maximum performance.

4.5.2 Plugin Architecture

The plugin architecture is a key component of the backend. The plugin architecture allows for easy integration of new connectors and transformation logic as additional data sources are incorporated into AIKO. The plugin architecture is designed to be extensible and flexible, allowing for easy integration with various data sources. The plugin architecture is stateless, allowing for easy scaling and deployment across cloud environments. The plugin architecture is designed to be highly scalable and fault-tolerant, ensuring that it can handle large volumes of data and requests efficiently.

The requirement for a new plugin to be incorporated into AIKO is that it must subclass an abstract class and define the necessary underlying methods. The abstract class is used as the interface into the plugin.

The LLMs are required to subclass the `LLMPlugin` class and reimplement the `_invoke` asynchronous method to provide a asynchronous generator. The async generator is used to asynchronously stream the response back to the frontend.

The DIS plugins are required to subclass the `FilePlugin` class and define a `_process_file` method. The `_process_file` takes a `FastAPI.UploadFile` and return the text extracted from the file.

The IPS is implemented as a fixed flow within the file upload route.

4.6 Backend Technologies and Frameworks

This section details the technologies, frameworks, toolkit and libraries crucial to the implementation of AIKO's backend.

Python [46] was used as the backend language of choice, due to ease of development and the tooling around the language. Version `3.12.6` was used as it is the latest stable version as of the development phase.

The backend of AIKO is hosted using Railway. NixPacks [47] are used to build containers for the backend, which can then be scaled horizontally.

4.6.1 🤗 Transformers

🤗 Transformers [5] (HuggingFace Transformers) provides APIs and tools to easily download and train state-of-the-art pretrained models. Using pretrained models can reduce your compute costs, carbon footprint, and save you the time and resources required to train a model from scratch. These models support common tasks in different modalities, such as:

- Natural Language Processing: text classification, named entity recognition, question answering, language modeling, summarization, translation, multiple choice, and text generation.
- Computer Vision: image classification, object detection, and segmentation.
- Audio: automatic speech recognition and audio classification.
- Multimodal: table question answering, optical character recognition, information extraction from scanned documents, video classification, and visual question answering.

Transformers support framework interoperability between PyTorch, TensorFlow [48], and JAX. This provides the flexibility to use a different framework at each stage of a model's life; train a model in three lines of code in one framework, and load it for inference in another. Models can also be exported to a format like ONNX and TorchScript for deployment in production environments.

🤗 Transformers was also used to provide the inference capabilities for LLMs both during development time as well as during server deployment.

4.6.2 Granian

Granian is a high-performance HTTP server for Python applications built on top of Rust's Hyper crate [49]. It provides a unified solution for serving Python web applications across multiple interfaces and protocols. Granian was designed to address common issues in Python web server stacks by providing a single, correct HTTP implementation with support for versions 1 and 2, avoiding the typical dependency composition of Gunicorn, Uvicorn, and HTTP tools on Unix systems. Key features:

- Support for multiple application interfaces: ASGI/3, RSGI, and WSGI
- Implements both HTTP/1 and HTTP/2 protocols
- Built-in HTTPS and WebSockets support
- Configurable worker processes and thread pooling
- Advanced backpressure mechanisms to optimize request handling
- Experimental support for free-threaded Python

Granian differentiates itself through its architecture, which runs a separate Rust runtime alongside the Python interpreter to handle I/O operations efficiently. This design allows it to maintain stable performance compared to other Python web servers. Granian provides flexible configuration options for workers, threads, and runtime modes (single-threaded or multi-threaded), making it adaptable to various deployment environments and application requirements. Unlike traditional Python web servers, Granian offers both process-based workers (with GIL) and thread-based workers (with free-threaded Python), giving developers options depending on their application's characteristics. It integrates with Python's standard logging module, making it compatible with existing logging configurations and practices. For applications that require high performance, protocol flexibility, and simplified deployment across platforms, Granian represents a modern alternative to the traditional Gunicorn/Uvicorn stack while maintaining compatibility with popular Python web frameworks.

4.6.3 FastAPI

FastAPI [50] is a modern, fast (high-performance), web framework for building APIs with Python based on standard Python type hints. It runs on the ASGI specification.

The key features are:

- Fast: Very high performance, on par with NodeJS and Go (thanks to Starlette and Pydantic [51]). One of the fastest Python frameworks available.
- Fast to code: Increase the speed to develop features by about 200% to 300%.
- Fewer bugs: Reduce about 40% of human (developer) induced errors.
- Intuitive: Great editor support. Completion everywhere. Less time debugging.
- Easy: Designed to be easy to use and learn. Less time reading docs.
- Short: Minimize code duplication. Multiple features from each parameter declaration. Fewer bugs.
- Robust: Get production-ready code. With automatic interactive documentation.
- Standards-based: Based on (and fully compatible with) the open standards for APIs: OpenAPI (previously known as Swagger) and JSON Schema.

FastAPI is used as the main backend web framework. It contains implementations for WebSockets and multipart responses, which were instrumental in the development of AIKO.

4.6.4 Pinecone

Pinecone [36] is a managed vector database designed for storing and querying high-dimensional vector embeddings. It enables fast similarity search for machine learning applications like recommendation systems, semantic search, and image retrieval.

Key features of Pinecone include:

- Scalable vector indexing and search
- Low-latency queries
- Support for metadata filtering
- Easy integration with popular ML frameworks

Pinecone allows developers to build AI-powered applications without having to manage complex infrastructure for vector search. It's used by companies across industries to deploy large-scale machine learning models in production.

Pinecone is used to host the embedded texts and perform retrieval on them. A seperate name space is created for every user providing secure seperation. The distance metric used was cosine.

4.6.5 Other Libraries and Frameworks

PyMuPDF [52] was used as the transformational logic behind PDF processing.

PyTest [53] and Molotov were used as testing frameworks.

`python-pptx` and `python-docx` provided insights into the processing and ingestion of `.docx` and `.pptx` respectively.

AIKO requires a GPU-enabled cloud platform, so our embedding models can be hosted and used. The models are hosted on Modal [54].

Modal is a cloud computing platform that specializes in simplifying the deployment and scaling of AI workloads. It enables developers to run any function in the cloud with minimal configuration, offering instant autoscaling for machine learning inference, data processing, and various computational tasks. The platform's core value proposition is its ability to transform local Python functions into cloud-deployed services with just a few lines of code.

Modal provides sub-second container starts through a custom Rust-based container stack, allowing for rapid iteration between local development and cloud deployment. Second, it offers zero-configuration deployments where hardware and container requirements are defined directly alongside Python functions. Third, Modal delivers seamless autoscaling capabilities, enabling applications to scale from zero to hundreds of GPUs within seconds based on demand.

The platform is particularly well-suited for AI-related workloads, including generative AI inference, fine-tuning, training, and batch processing. It provides immediate access to high-performance computing resources such as NVIDIA A100 and H100 GPUs without waiting for provisioning. Modal also supports various

storage solutions, job scheduling, web endpoints, and built-in debugging tools. AIKO uses the `L40S` GPU from NVIDIA, which contains 48GB of VRAM.

4.6.6 NLP-Enabled Content Processor

One core aspect of AIKO is the Content Processor. It was implemented using one of the latest LLMs. The LLMs are finetuned to efficently "remember" the information in the text without any output tokens. Once all of the entity has been parsed, the LLM output is then structured using a completions endpoint (rather than a chat endpoint) to reply only with JSON. LLMs are few shot learners [23], and as such a few examples of the intended outputs are given alongside the system prompt.

Intitially, `llama3-8b-8192` was used, which had a context window of 8192 tokens. The model was later updated to a finetuned version of `mistral-8x7b-32768` which improved the context window to 32768 tokens. The model was then further upgraded to a instruction-finetuned model from Cohere [55] called `command-r-plus` which significantly increased the context window to 128,000 tokens. A proprietary model from Anthropic `claude-3.7-sonnet` is also used as part of a visual processing pipeline.

The content ingestion is taken care by various multimodal LLMs and Computer Vision Models. Currently Anthropic's Claude 3.7 Sonnet Model as well as model from Mistral called Pixtral-12B is being used for images. These models convert the image into an Intermediate Representation, which is a common, extensible format to facilitate seamless integration with downstream services and processes.

The audio is processed by virtue of a Large-Scale Weak Supervision model called Whisper Large v3 from OpenAI [56].

Other entites (such as text documents) are split into their constituents and then the components are processed.

4.6.7 Search Engine

The search engine is designed using two parallel workflows.

A embedding flow that takes the user's query, modifies it and embeds it to get a vector representation that resembles the target entity.

This vector is then used to do a Nearest Neighbor seach over the corpus of the user's data. The vector search is done inside Pinecone.

Another flow takes the query, extracts the keywords from the query and does a Fuzzy Search over the corpus to find related fragments from the corpus.

The results from both are then reranked to build a final result list.

```
%% width: 60%
%%! Diagrammatical Representation of the Search Routine
graph TD
    A[User Query] --> B[Embedding Flow]
    A --> C[Fuzzy Search]
    B --> D[Nearest Neighbor Search]
    C --> E[ReRanking]
    D --> E
    E --> F[Final Results]
```

4.7 Frontend Development

The frontend was designed using the latest web frameworks to ensure performance and ease of development.

The frontend is the main interface for common users. It allows the following functions:

- Viewing Entities ingested into the system.
- Searching for Entities using Neural Search.
- Ingestion of new Entities into AIKO's
- Q/A chat with the Entities regardless of their modality.

4.7.1 User Interface and Experience Design

AIKO's user interface has been designed to be intuitive, responsive and accessible across various devices. It follows standardized guidelines on UI Design and emphasizes simplicity and effectiveness [57]

The core user interface is modelled after an utilitarian design, with a focus on functionality and ease of use. The design is minimalistic, with a clean and uncluttered layout that prioritizes the main content. The color scheme is neutral, with subtle accents to highlight important elements. The typography

is clear and legible, ensuring easy readability. The application is structured in a way that maximizes the screen real estate for the main content area.

The following section details the technicalities of the frontend, the actual design as well as a project walkthrough can be found in Section 5.1

4.8 Frontend Technologies and Frameworks

This section details the technologies, frameworks, toolkit and libraries crucial to the implementation of AIKO's frontend.

TypeScript was used as the frontend language of choice. Svelte was used as the frontend framework. The tooling for the frontend is handled using vite.

The frontend (and the serverless part of the frontend) is hosted on Vercel [58].

4.8.1 TypeScript

TypeScript [59, 60] is a programming language that builds upon JavaScript by adding optional static typing and other features.

TypeScript is:

- A superset of JavaScript
- Developed and maintained by Microsoft
- Designed to make development of large-scale applications easier

Key features:

- Static typing
- Object-oriented programming features
- Improved tooling and IDE support
- Compatibility with existing JavaScript code

TypeScript compiles to plain JavaScript, allowing it to run in any environment that supports JS. Typescript was used as it provides type safety and better frontend development experience.

4.8.2 Svelte

Svelte is a modern front-end framework that takes a different approach to building user interfaces compared to traditional frameworks like React and Vue. Instead of relying on a virtual DOM for rendering, Svelte

compiles your components into highly optimized JavaScript code at build time. This eliminates the need for expensive DOM manipulations during run-time, resulting in better performance and potentially smaller bundle sizes.

Key Features:

- Declarative Syntax: Svelte uses a declarative syntax, making it easy to define the structure and behavior of your UI components.
- Component-Based Architecture: Svelte promotes a component-based approach, allowing you to break down your application into reusable and modular components.
- Reactivity: Svelte's reactivity system automatically updates the UI when data changes, ensuring your application stays in sync with the underlying state.
- No Virtual DOM: Svelte compiles components directly to efficient JavaScript, eliminating the overhead of a virtual DOM.
- Strong Community and Ecosystem: Svelte has a growing community and ecosystem, with a wide range of tools, libraries, and resources available.

Svelte 5 introduces a concept called "runes", which are a powerful set of primitives for controlling reactivity. Runes significantly improve the performance and allow for finegrained reactivity.

SvelteKit [61] is a full-stack framework built on top of Svelte. It provides a set of tools and conventions for building server-rendered web applications, offering a more streamlined development experience.

Key Features:

- Server-Side Rendering (SSR): SvelteKit renders your components on the server, improving initial page load performance and SEO.
- Code Splitting: SvelteKit automatically splits your code into smaller bundles, reducing the initial load time of your application.
- Routing: SvelteKit provides a built-in routing system for managing navigation within your application.
- Data Fetching: SvelteKit offers built-in mechanisms for fetching data on the server or client-side, depending on your needs.

- File-Based Routing: SvelteKit uses a file-based routing convention, making it easy to organize and manage your application's routes.
- Static Site Generation (SSG): SvelteKit can generate static versions of your application, which can be hosted on a CDN for maximum performance.

In addition, libraries like `svelte-motion`, `iconify-svelte`, `svelte-markdown` and `svelte-sonner` were used to provide additional functionalities.

4.8.3 Vite

Vite is a modern front-end build tool that aims to provide a fast and efficient development experience. It leverages native ES modules for development, eliminating the need for bundling during the development process. This results in significantly faster hot module replacement (HMR) and overall build times.

Key Features:

- Native ES Modules: Vite takes advantage of the browser's native ES module support to serve source files directly, eliminating the need for bundling during development.
- Fast HMR: Vite's efficient HMR implementation allows for near-instantaneous updates to your application, improving your development workflow.
- No Bundling During Development: By avoiding bundling during development, Vite achieves significant performance gains, especially for larger projects.
- Production Bundling: Vite uses Rollup under the hood for production builds, ensuring that your application is optimized for deployment.
- Rich Plugin Ecosystem: Vite's plugin system allows you to extend its functionality with various tools and features, such as CSS preprocessors, linters, and more.

Vite uses a plugin architecture and as such requires a plugin for Svelte. The plugin is called `vite-plugin-svelte`.

Unit Testing for the frontend is handled in a program called Vitest [62].

4.8.4 Tailwind CSS

Tailwind CSS [63] is a utility-first CSS framework that provides a set of pre-defined CSS classes for styling HTML elements. Unlike traditional

CSS frameworks that provide pre-built components, Tailwind CSS empowers developers to build custom user interfaces by combining these utility classes.

Key Features:

- Utility-first approach: Tailwind CSS focuses on providing low-level utility classes that can be combined to create custom styles. This approach gives developers greater control over their designs and reduces the need for custom CSS.
- Customizable: Tailwind CSS allows developers to customize the framework's default styles and add their own utility classes, providing flexibility for different project requirements.
- Responsive design: Tailwind CSS provides a responsive design system with built-in classes for creating responsive layouts and styles.
- Dark mode support: Tailwind CSS includes classes for creating dark mode themes, making it easy to design websites that adapt to different lighting conditions.
- Large community and ecosystem: Tailwind CSS has a large and active community with numerous resources, plugins, and tools available to enhance its functionality and streamline development.

Additionally daisyUI was used to provide a set of customizable prebuilt components that could be used in the frontend.

4.8.5 Prettier

Prettier is an opinionated code formatter that automatically formats your code according to a set of predefined rules. It helps maintain consistent code style across your project and reduces the need for manual formatting.

4.8.6 Playwright

Playwright [64] is a versatile and efficient testing framework designed to automate web applications across different browsers, platforms, and languages. It offers a wide range of features and benefits that make it a popular choice for modern web development teams.

Key Features:

- Cross-browser compatibility: Supports Chromium, WebKit, and Firefox, ensuring your tests work consistently across different browsers.
- Cross-platform support: Runs on Windows, Linux, and macOS, allowing you to test on various operating systems.
- Multiple languages: Offers APIs for TypeScript, JavaScript, Python, .NET, and Java, catering to diverse programming preferences.
- Mobile web testing: Emulates Google Chrome for Android and Mobile Safari, enabling testing on mobile devices.
- Resilient testing: Employs auto-waiting and web-first assertions to reduce flakiness and improve test reliability.
- Tracing and debugging: Captures execution traces, videos, and screenshots for in-depth analysis and troubleshooting.
- Full isolation: Creates browser contexts for each test, ensuring independent execution and avoiding interference.
- Fast execution: Leverages efficient browser context creation and log-in state caching to speed up test runs.
- Powerful tooling: Includes code generation, Playwright inspector, and Trace Viewer for enhanced development and debugging.

Playwright was used to implement End-to-End tests for AIKO.

4.8.7 Other Libraries and Frameworks

- `autoprefixer`: A PostCSS plugin that automatically adds vendor prefixes to CSS rules, ensuring cross-browser compatibility.
- `postcss`: A tool for transforming CSS with JavaScript plugins, enabling advanced CSS processing and optimization.
- `tailwind-merge`: A Tailwind CSS plugin that merges utility classes to reduce duplication and optimize CSS output.
- `class-variance-authority`: CVA is a versatile and efficient library designed to simplify the creation and management of CSS variants in TypeScript projects. It offers

a clean and intuitive API that allows you to define variants based on props, ensuring type safety and reducing the risk of errors.

- `clsx`: A tiny utility for constructing className strings conditionally.

4.9 Web Extension Development

The AIKO Web Extension was developed to allow users to save web pages and YouTube videos directly into AIKO. It is written using TypeScript, and uses the WebExtension API to interact with the browser. The extension is written in such a way that it can be installed on both Chrome and Firefox browsers. The extension authenticates and communicates with the AIKO backend directly.

The extension is self-distributed for both Chrome and Firefox. The extension is not installable from the Chrome Web Store or the Firefox Add-ons Store. Links to install and update along with the instructions are provided in both, the documentation as well as the hosted instance of AIKO.

The extension uses the following technologies and frameworks:

- `isomorphic-dompurify`: A library for sanitizing HTML content to prevent XSS attacks.
- `notyf`: A library for displaying toast notifications in the browser.
- `@mozilla/readability`: A library for extracting the main content from a web page.
- `webextension-polyfill`: A polyfill for the WebExtension API, providing a consistent interface across different browsers.

Additionally, a vite plugin was written from scratch to implement the build pipeline required for the extension. The code can be found in the `src-extension` folder of the repository with the filename `manifestPlugin.js`

CHAPTER 5
RESULTS AND DISCUSSIONS

This chapter highlights the results obtained from the implementation, deployment and testing of AIKO while providing a user's perspective of the system.

5.1 Project Walkthrough

This section focuses on the features and functionalities of AIKO as presented to a user. A detailed walkthrough with screenshots and explanations is provided to give a comprehensive understanding of the system's capabilities.

5.1.1 Landing Page

The landing page is the first point of contact for users visiting AIKO. It provides an overview of the system's features and functionalities, guiding users on how to get started. The landing page includes a brief description of AIKO, key benefits, and a call-to-action to sign up or log in.

The landing page allows users to move to the authentication page or scroll down to learn more about AIKO's features.

5.1.2 Authentication Page

AIKO has a simple and intuitive authentication page that allows users to sign up or log in to the system. Users can create an account by providing their email address and password. An option to signin using Google and GitHub is also provided. The authentication page includes a link to the documentation page as well as a support email.

Once logged in, users are redirected to the dashboard.

5.1.3 Dashboard

The dashboard is the central hub of AIKO, providing users with an overview of their uploaded entities, search functionality, and other key features. The dashboard includes a naviation bar and a paginated entity list.

Users can utilize the naviation bar to access the various pages AIKO has to offer. The links in order are

1. Dashboard (Subsection 5.1.3)

2. Profile (Subsection 5.1.4)
3. Add File
4. Search
5. Chat

This navigation bar is present on all pages of the application.

A left and right sidebar is present on all pages of the application, the left sidebar houses quick actions, while the right sidebar houses AIKO's Model Parameter selectors.

The following parameters can be tuned:

- `temperature`: The temperature of the model. Higher temperatures lead to more randomness in the output.
- `top_k`: The number of entities to consider for contextual Q/A.
- `model_choice`: The model to use for the Q/A.

The bottom right corner houses the status icon, which shows the current status of the system. The status icon can be one of the following:

- `green`: The system is operational.
- `yellow`: The system is in an uncertain state, it will resolve to either green or red.
- `red`: The system is down.

The dashboard also includes a paginated entity list that displays the user's entities. Each entity in the list includes a title, description, and a set of actions that users can perform on the entities. Users can view, edit, delete, and download entities from the entity list.

5.1.4 Profile Page

The profile page contains basic info about the user, it serves no functional purpose other than giving the user their own information.

5.1.5 Add File Page

The Add File Page allows users to upload entities into AIKO. The page includes a file upload form, where users can select a file to upload. The supported file formats are `text/*`, `application/pdf`,`.docx`, `.pptx`, `image/*`, and `audio/*`.

Upon selection of a file to upload, the UI changes to show the `plugin_choice`. This allows the user to select the plugin to use for the entities processing phase. Each plugin is different, and one may suit the user better than the other. The plugin may require further config parameters, which are shown as choices to the user.

Once the requsite parameters are selected, the user can click on the `Upload` button to upload the file. The file is then processed by the backend and the user is shown a success page.

The user can either choose to upload another file or view the entity just uploaded.

5.1.6 Search Page

The search page encapsulates the core feature of AIKO, the neural search. The search page allows users to enter a query and retrieve relevant entities from the system. The search page includes a search bar, where users can enter their query, and a search button to initiate the search.

The search results are displayed in a paginated list, similar to the entity list on the dashboard. Each search result includes a title, description, and a set of actions that users can perform on the entity. Users can view, download, or chat with the entity from the search results.

5.1.7 Entity View Page

The entity view page allows users to view the details of a specific entity. The page includes the entity's title, description, and content. Users can also view the metadata associated with the entity, such as the date of creation, and file type.

The page allows users to preview the entity, download the entity, or chat with the entity.

The user is also allowed to delete the entity from here. The user is shown a confirmation dialog before the entity is deleted.

5.1.8 Chat Page

Clicking on the chat button, the user is taken to the chat page. The chat page allows users to interact with the entity using natural language. The chat page includes a chat window, where users can enter their queries and receive responses from the entity.

The left panel shows the context of the entity, while the right panel shows the chat window. Entering a query into the chat window and submitting it, the user is shown the response from the entity. Finally, the user can logout using the button given in the left sidebar, which navigates into the authentication page.

5.2 Results

The implementation and deployment of AIKO provided the following results:

5.2.1 Unified Personalized Knowledge Repository

AIKO was able to provide a unified repository for users to store and manage their knowledge. AIKO allows all users to upload and manage their documents, images, and audio files in a single location.

5.2.2 Enhanced Information Retrieval System

AIKO was able to enhance the information retrieval system by providing a neural search engine that allows users to search for documents using natural language queries. The search engine uses state-of-the-art language models to provide accurate and relevant search results.

5.2.3 Improved Knowledge Management Capabilities

The overall time spent on searching for information was reduced due to the improved search capabilities of AIKO. Users were able to quickly find the information needed, leading to increased productivity and efficiency.

5.2.4 Documentation and Self-Hostable Instance

AIKO was created with the intention of being self-hostable using any cloud provider. The documentation contains instructions on how to deploy AIKO on a cloud provider of choice. The self-hostable instance allows users to have complete control over their data and the system. Incase a user has a hardware system capable of running state-of-the-art models, AIKO can be run on such a local machine, without the need for an internet connection at all.

CHAPTER 6
CONCLUSION

6.1 Summary

Through AIKO, users can store, manage, and retrieve information across various modalities, including text, images, and audio. The system provides a unified repository for users to organize their knowledge and access it efficiently. The neural search engine and chat feature enhance the information retrieval process, allowing users to find relevant information quickly and easily. AIKO's self-hostable instance provides users with complete control over their data and the system, ensuring data privacy and security.

- AIKO provides a comprehensive solution for knowledge management, offering users a unified platform to store and retrieve information across different modalities. Potential use cases include academic, research, personal and corporate use as a knowledge bank.
- The neural search engine and chat feature enhance the user experience, enabling users to find information quickly and efficiently.
- AIKO was able to provide a unified repository for users to store and manage their personal knowledge regardless of modalities.
- AIKO was able to reduce the time spent on searching for information, improving user productivity and efficiency.
- AIKO was able to understand context and nuances in user queries, providing accurate and relevant search results.

6.2 Conclusions

AIKO establishes itself as a significant advancement in the field of knowledge management and retrieval systems, successfully integrating multiple modalities while maintaining user privacy through self-hosting capablities. The system demonstrated the practical application of modern AI and ML technologies in solving real-world information management challenges. The implementation of

a neural search and natural language processing capablities has proven effective in understanding and retrieving relevant information across different content types.

6.3 Future Work

AIKO has the potential to be further developed and expanded to include additional features and capabilities. Future work could focus on the following areas:

- Increased Modality Support

  AIKO has the potential to be extended to allow the ingest, processing and search of newer file formats as the file formats become available. Recent developments have made it possible to input videos into LLMs [65].

- Better Self-Hosting Capabilities

  Current version of AIKO is self-hostable but uses the cloud for processing. Future revisions of AIKO could be made to be completely self-hosted, allowing users to run the entire system on a local machine, without the need for an internet connection.

- Better Explanablity

  AIKO in its current state is pretty transparent about the routes the data takes, However the decision making process of the AI is not always transparent to the user. Future work could focus on improving the explainability of AIKO's decision-making process, providing users with more transparency into how the system arrives at its conclusions.

REFERENCES

[1] Mem - the AI notes app that keeps you organized. Retrieved July 14, 2024 from https://get.mem.ai/

[2] Brain assistant. Retrieved July 14, 2024 from https://mybrain.zone/dashboard

[3] Vedant Parikh, Vidit Mathur, Parth Mehta, Namita Mittal, and Prasenjit Majumder. 2021. LawSum: A weakly supervised approach for Indian Legal Document Summarization. Retrieved from https://arxiv.org/abs/2110.01188

[4] AI Brain Bank. Retrieved July 14, 2024 from https://aibrainbank.com/

[5] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. HuggingFace's Transformers: State-of-the-art Natural Language Processing. Retrieved July 14, 2024 from https://arxiv.org/abs/1910.03771

[6] iWeaver AI. 2024. Remember Recall Reuse your knowledge | iWeaver AI Memory Tool. Retrieved July 14, 2024 from https://www.iweaver.ai/

[7] Personal knowledge AI | Keepi. Retrieved July 14, 2024 from https://www.keepi.ai/

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. *arXiv (Cornell University)* 30, (2017), 5998–6008. Retrieved July 14, 2024 from https://arxiv.org/pdf/1706.03762v5

[9] Yujia Zhou, Zheng Liu, and Zhicheng Dou. 2024. AssistRAG: Boosting the Potential of Large Language Models with an Intelligent Information Assistant. (2024). Retrieved from https://arxiv.org/abs/2411.06805

[10] Anantha Sharma, Sheeba Elizabeth John, Fatemeh Rezapoor Nikroo, Krupali Bhatt, Mrunal Zambre, and Aditi Wikhe. 2024. Mitigating Hallucination with

ZeroG: An Advanced Knowledge Management Engine. (2024). Retrieved from https://arxiv.org/abs/2411.05936

[11] Vamsi Krishna Kommineni, Birgitta König-Ries, and Sheeba Samuel. 2024. Harnessing multiple LLMs for Information Retrieval: A case study on Deep Learning methodologies in Biodiversity publications. (2024). Retrieved from https://arxiv.org/abs/2411.09269

[12] Zhonghan Chen, Ruiyuan Zhang, Xi Zhao, Xiaojun Cheng, and Xiaofang Zhou. 2024. Exploring the meaningfulness of Nearest Neighbor Search in High-Dimensional Space. *arXiv (Cornell University)* (October 2024). https://doi.org/10.48550/arxiv.2410.05752

[13] Mohammad Kachuee, Sarthak Ahuja, Vaibhav Kumar, Puyang Xu, and Xiaohu Liu. 2024. Improving tool retrieval by leveraging large language models for query generation. *arXiv (Cornell University)* (November 2024). https://doi.org/10.48550/arxiv.2412.03573

[14] Pranav Narayanan Venkit, Philippe Laban, Yilun Zhou, Yixin Mao, and Chien-Sheng Wu. 2024. Search engines in an AI era: The false promise of factual and verifiable Source-Cited responses. *arXiv (Cornell University)* (October 2024). https://doi.org/10.48550/arxiv.2410.22349

[15] David Dukić, Marin Petričević, Sven Ćurković, and Jan Šnajder. 2024. TakeLab Retriever: AI-Driven Search Engine for Articles from Croatian News Outlets. *arXiv (Cornell University)* (November 2024). https://doi.org/10.48550/arxiv.2411.19718

[16] Guangzhi Xiong, Eric Xie, Amir Hassan Shariatmadari, Sikun Guo, Stefan Bekiranov, and Aidong Zhang. 2024. Improving Scientific Hypothesis Generation with Knowledge Grounded Large Language Models. (2024). Retrieved from https://arxiv.org/abs/2411.02382

[17] Nicholas J. Radcliffe and Patrick D. Surry. 1995. *Fundamental limitations on search algorithms: Evolutionary computing in perspective*. https://doi.org/10.1007/bfb0015249

[18] Guangxin He, Zonghong Dai, Jiangcheng Zhu, Binqiang Zhao, Chenyue Li, You Peng, Chen Wang, and Binhang Yuan. 2024. Zero-Indexing Internet Search augmented generation for large language models. *arXiv (Cornell University)* (November 2024). https://doi.org/10.48550/arxiv.2411.19478

[19] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference. Retrieved from https://arxiv.org/abs/2403.04132

[20] 2012. *The OAUTH 2.0 Authorization Framework*. https://doi.org/10.17487/rfc6749

[21] XiPeng Qiu, TianXiang Sun, YiGe Xu, YunFan Shao, Ning Dai, and XuanJing Huang. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences* 63, 10 (September 2020), 1872–1897. https://doi.org/10.1007/s11431-020-1647-3

[22] Welch Labs. 2024. The moment we stopped understanding AI [AlexNet]. Retrieved September 26, 2024 from https://www.youtube.com/watch?v=UZDiGooFs54

[23] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *Neural Information Processing Systems* 33, (2020), 1877–1901. Retrieved July 14, 2024 from https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[24] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. Retrieved from https://arxiv.org/abs/1301.3781

[25] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, October 2014. Association for Computational Linguistics, Doha, Qatar, 1532–1543. https://doi.org/10.3115/v1/D14-1162

[26] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Retrieved from https://arxiv.org/abs/1810.04805

[27] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9, 86 (2008), 2579–2605. Retrieved from http://jmlr.org/papers/v9/vandermaaten08a.html

[28] Mark A. Kramer. 1991. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal* 37, 2 (1991), 233–243. https://doi.org/https://doi.org/10.1002/aic.690370209

[29] Leland McInnes, John Healy, and James Melville. 2020. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. (2020). Retrieved from https://arxiv.org/abs/1802.03426

[30] Avivit Levy, B. Riva Shalom, and Michal Chalamish. 2024. A Guide to Similarity Measures. (2024). Retrieved from https://arxiv.org/abs/2408.07706

[31] Michael W. Berry, Zlatko Drmac, and Elizabeth R. Jessup. 1999. Matrices, Vector Spaces, and Information Retrieval. *SIAM Review* 41, 2 (1999), 335–362. https://doi.org/10.1137/S0036144598347035

[32] Bijoyan Das and Sarit Chakraborty. 2018. An Improved Text Sentiment Classification Model Using TF-IDF and Next Word Negation. Retrieved from https://arxiv.org/abs/1806.06407

[33] Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Haonan Chen, Zheng Liu, Zhicheng Dou, and Ji-Rong Wen. 2024. Large Language Models for Information Retrieval: A Survey. (2024). Retrieved from https://arxiv.org/abs/2308.07107

[34] Donna Harman. 2011. *Information retrieval evaluation*. https://doi.org/10.1007/978-3-031-02276-0

[35] David Maier. 1983. *The theory of relational databases*. Retrieved from http://ci.nii.ac.jp/ncid/BA00959711

[36] The vector database to build knowledgeable AI | Pinecone. Retrieved July 14, 2024 from https://www.pinecone.io/

[37] Kai Petersen, Claes Wohlin, and Dejan Baca. 2009. *The waterfall model in Large-Scale development*. https://doi.org/10.1007/978-3-642-02152-7_29

[38] Devansh Parapalli, Kaustubh Warade, Aditya Deshmukh, and Yashasvi Thool. 2024. Software Requirement Specification for AIKO (Version 1.2 approved). Retrieved July 31, 2024 from https://aiko.parapalli.dev/docs/static/srs.pdf

[39] Devansh Parapalli, Kaustubh Warade, Aditya Deshmukh, and Yashasvi Thool. 2024. Software Design Document for AIKO (Version 1.5 approved). Retrieved August 16, 2024 from https://aiko.parapalli.dev/docs/static/sdd.pdf

[40] Kief Morris. 2016. *Infrastructure as code: Managing servers in the cloud*. Retrieved from https://www.amazon.com/Infrastructure-Code-Managing-Servers-Cloud/dp/1491924357

[41] Devansh Parapalli, Kaustubh Warade, Aditya Deshmukh, and Yashasvi Thool. 2024. Verification and Validation Document for AIKO (Version 1.0). Retrieved September 17, 2024 from https://aiko.parapalli.dev/docs/static/vnv.pdf

[42] Devansh Parapalli, Kaustubh Warade, Aditya Deshmukh, and Yashasvi Thool. 2024. Proposed Budget for AIKO (Version 1.2). Retrieved August 7, 2024 from https://aiko.parapalli.dev/docs/static/budget.pdf

[43] Supabase | the open source Firebase alternative. Retrieved July 14, 2024 from https://supabase.com/

[44] 2024. PostgreSQL. Retrieved July 14, 2024 from https://www.postgresql.org/

[45] Harrison Chase. 2022. LangChain. Retrieved from https://github.com/langchain-ai/langchain

[46] 2024. Welcome to Python.org. Retrieved July 14, 2024 from https://python.org/

[47] Getting started | Nixpacks. Retrieved July 14, 2024 from https://nixpacks.com/

[48] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow, Large-scale machine learning on heterogeneous systems. https://doi.org/10.5281/zenodo.4724125

[49] . 2025. GitHub - emmett-framework/granian: A Rust HTTP server for Python applications — github.com.

[50] Malhar Lathkar. 2023. *High-Performance Web Apps with FastAPI*. https://doi.org/10.1007/978-1-4842-9178-8

[51] Welcome to Pydantic - Pydantic. Retrieved July 14, 2024 from https://docs.pydantic.dev/latest/

[52] Artifex. PyMuPDF 1.24.7 documentation. Retrieved July 14, 2024 from https://pymupdf.readthedocs.io/en/latest/

[53] Holger Krekel, Bruno Oliveira, Ronny Pfannschmidt, Floris Bruynooghe, Brianna Laugher, and Florian Bruhin. 2004. pytest x.y. Retrieved July 14, 2024 from https://github.com/pytest-dev/pytest

[54] Modal: High-performance AI infrastructure. Retrieved from https://modal.com/

[55] Cohere | The leading AI platform for enterprise. Retrieved July 14, 2024 from https://cohere.com/

[56] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2022. Robust Speech Recognition via Large-Scale Weak Supervision. (2022). Retrieved from https://arxiv.org/abs/2212.04356

[57] Jakob Nielsen, Kelly Gordan, Kate Morgan, and Feifei Liu. 2024. 10 Usability Heuristics for User Interface Design. Retrieved July 28, 2024 from https://www.nngroup.com/articles/ten-usability-heuristics/

[58] Vercel: Build and deploy the best web experiences with the Frontend Cloud. Retrieved July 14, 2024 from https://vercel.com/

[59] Gavin Bierman, Martín Abadi, and Mads Torgersen. 2014. *Understanding TypeScript*. https://doi.org/10.1007/978-3-662-44202-9_11

[60] Gavin Bierman, Mart\in Abadi, and Mads Torgersen. 2014. Understanding typescript. In *European Conference on Object-Oriented Programming*, 2014. 257–281.

[61] SvelteKit. Retrieved July 14, 2024 from https://kit.svelte.dev/

[62] Anthony Fu, Matias Capeletto, and Vitest contributors. 2021. Vitest: Next Generation Testing Framework. Retrieved from https://vitest.dev/

[63] Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. Retrieved July 14, 2024 from https://tailwindcss.com/

[64] Microsoft Corporation. 2021. Playwright: Any browser • Any platform • One API. Retrieved from https://playwright.dev/

[65] Qwen-Team. 2024. Qwen2-VL. (2024).

Appendix A: LLM System Prompts

Image Description Prompt

```
You are a crucial component of AIKO, an advanced AI-powered Knowledge
Organizer.
Your primary function is to analyze images with extreme precision and
translate visual information into detailed, accurate textual descriptions.

Core Responsibilities:
1. Provide comprehensive, highly detailed descriptions of all image contents.
2. Ensure utmost accuracy in all observations and descriptions.
3. Transcribe all visible text verbatim, regardless of its location or
significance within the image.
4. Describe all objects, scenes, and notable elements with maximum detail.

Key Guidelines:
- Accuracy is paramount. Do not speculate or infer beyond what is clearly
visible.
- Describe the image systematically, from overall composition to minute
details.
- For documents, outline the structure and content, highlighting key
information.
- Identify public figures or celebrities if clearly recognizable, but do not
attempt to name private individuals.
- Include relevant contextual information (e.g., time of day, weather
conditions, apparent location) when discernible.
- Note the quality, style, or unique characteristics of the image (e.g., black
and white, cartoon, digital rendering).
- Describe colors, textures, patterns, and materials where applicable.
- Mention any visible brands, logos, or identifiable products.
- For charts, graphs, or infographics, explain the data representation and key
insights.

Output Format:
1. Image Overview: Briefly summarize the main subject or theme of the image.
2. Detailed Description: Systematically describe all elements of the image.
3. Text Transcription: List all visible text, organized by location or
relevance.
```

```
4. Notable Elements: Highlight any unique, unusual, or particularly
significant aspects.
5. Context and Interpretation: Provide any relevant contextual information or
apparent purpose of the image.

Remember: Your role is to be the eyes of the system. Describe everything you
see with precision, leaving no detail unmentioned, no matter how small it may
seem.
Your description should enable someone who cannot see the image to form a
complete and accurate mental picture of its contents.
```

Summary, Metadata, and Tags Generation Prompt

```
You are an efficient content tagger and summarizer. Your task is to create
tags, summary and title for any given piece of text.
The tags must be relevant, informative and contain identifying information
about the text. It can contain proper nouns, keywords, etc.
Incase any specific identifier is present in the text, such as the publisher
name or the author name, it should be included in the tags.
The tags must be in lowercase with space replaced by '-', presented as a JSON
list. Do not use any non alphanumeric characters, except '-',  in the tags.
The summary must be very concise and informative.
The title should be concise and short. It must be a single line of text. It
can include the type of document as well, such as Resume, Article, etc.
The JSON format for your output is as follows:
{
    "summary": "summary",
    "title": "title",
    "tags": ["tag1", "tag2", "tag3"]
}
Do not output anything other than the JSON object.
```

The output was further structured by restricting the LLM completion to begin with the `{` character. The output was then dirty parsed using the following code:

```python
import json
import re

def fix_broken_escapes(text):
    def replace_escape(match):
        escaped_char = match.group(1)
        if escaped_char in "nrt\"'":
```

```
            return "\\" + escaped_char
        return escaped_char

    pattern = r'\\([^\\nrt"\'])'
    return re.sub(pattern, replace_escape, text)


def dirty_json_parser(input_string):
    # Remove leading and trailing whitespace, and fix broken escape sequences
    input_string = fix_broken_escapes(input_string.strip())

    # Try to find the start of the JSON object or array
    start_match = re.search(r"[{\[]", input_string)
    if not start_match:
        raise ValueError("No JSON object or array found in the input string")

    start_index = start_match.start()

    # Find the matching closing bracket or brace
    stack = []
    end_index = -1
    in_string = False
    escape_next = False

    for i, char in enumerate(input_string[start_index:], start=start_index):
        if not in_string:
            if char in "{[":
                stack.append(char)
            elif char in "}]":
                if not stack:
                    raise ValueError("Unmatched closing bracket or brace")
                if (char == "}" and stack[-1] == "{") or (
                    char == "]" and stack[-1] == "["
                ):
                    stack.pop()
                    if not stack:
                        end_index = i + 1
                        break
                else:
                    raise ValueError("Mismatched brackets or braces")
```

```
    if char == '"' and not escape_next:
        in_string = not in_string

    escape_next = char == "\\" and not escape_next

# If we didn't find a closing bracket/brace, assume it's at the end
if end_index == -1:
    end_index = len(input_string)

# Extract the JSON substring
json_string = input_string[start_index:end_index]

# Add missing closing brackets/braces
while stack:
    if stack[-1] == "{":
        json_string += "}"
    elif stack[-1] == "[":
        json_string += "]"
    stack.pop()

# Try to parse the extracted JSON
try:
    parsed_json = json.loads(json_string)
    return parsed_json
except json.JSONDecodeError as e:
    raise ValueError(f"Invalid JSON: {e}")
```

Appendix B: Database Schema