RAPTOR
Recurssive Abstractive Processing for Tree Organized Retreival using PHI-4 LLM by Microsoft

A PROJECT REPORT
SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE COMPLETION OF CS4200-MAJOR PROJECT

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

SUBMITTED BY

Vishnu Vardhan Reddy Kandada
(Enrollment No. 21CS002383)

FACULTY OF COMPUTING AND INFORMATIC
SIR PADAMPAT SINGHANIA UNIVERSITY
UDAIPUR 313601, INDIA

JAN, 2025

RAPTOR
Recurssive Abstractive Processing for Tree Organized Retreival using PHI LLM by Microsoft

a Project Report
Submitted in partial fulfillment of the requirements
for CS4200-Major Project

BACHELOR OF TECHNOLOGY
in
Computer Science & Engineering

submitted by

Vishnu Vardhan Reddy Kandada
(Enrollment No. 21CS002383)

Under the guidance of
Prof. Alok Kumar
(Project Coordinator)
and
Dr. Utsav Upadyay
(Supervisor)

FACULTY OF COMPUTING AND INFORMATICS
SIR PADAMPAT SINGHANIA UNIVERSITY
UDAIPUR 313601, India

JAN, 2025

Faculty of Computing and Informatics
Sir Padampat Singhania University
Udaipur, 313601, India

CERTIFICATE

I, **V**ishnu Vardhan Reddy Kandada, hereby declare that the work presented in this project report entitled **RAPTOR, Recurssive Abstractive Processing for Tree Organized Retreival using PHI-4 LLM by Microsoft** for the completion of CS4200-Major Project and submitted in the **Faculty of Computing and Informatics** of the **Sir Padampat Singhania University, Udaipur** is an authentic record of my own work carried out under the supervision of **Prof. Alok Kumar, Professor**, and **Dr. Utsav Upadyay, Professor**. The work presented in this report has not been submitted by me anywhere else.

Vishnu Vardhan Reddy Kandada
21CS002383

This is to certify that the above statement made by the candidate is true to the best of my knowledge and belief.

Prof. Alok Kumar
Professor
Project Coordinator

Dr. Utsav Upadyay
Professor
Supervisor

Place: Udaipur
25-01-2025

Acknowledgements

Inscribing these words of gratitude feels akin to painting a masterpiece on the canvas of appreciation. This incredible path of learning and exploration would not have been possible without the unflinching support and encouragement of the great individuals who have paved the road for my accomplishment.

I reserve a special place in my heart for my beloved parents, whose unwavering love, unwavering support, and unwavering belief in my abilities have been the bedrock upon which my dreams have flourished. Their persistent support, sacrifices, and unshakable trust in my abilities have been the driving factors behind my quest for knowledge and academic pursuits.

First and foremost, I owe a tremendous debt of gratitude to my esteemed supervisor, **Dr. Alok Kumar** , whose guidance and advice have been the compass guiding me through the many twists and turns of this thesis. His stimulating conversations, insightful feedback, kind advice, and boundless forbearance have challenged me to push the boundaries of my capabilities and inspired me to strive for academic excellence. I am very thankful for the trust you put in me and the chances you gave me to grow both professionally and personally. I am grateful beyond words for the opportunity to have worked under your guidance, and I hope my thesis serves as a fitting tribute to your hard work, knowledge, and encouragement.

I like to thank **Dr. Utsav Upadyay** , Designation, Computer Science and Engineering Department, and **Dr. Alok Kumar** , Head of the Department, Computer Science and Engineering Department, for their extended support.

I would like to extend a heartfelt thank you to, **Mr. Sai Dev Pappu**, **Mr. Shri Hari Gudepally**, **Mr. Gowtham Reddy Gudla**, **Mr. Nikhil Cheryala** my incredible classmates and friends, who have been a constant source of support, camaraderie, and inspiration. Their presence has made the often-trying process of writing a thesis into one that is filled with joy and fun. Finally, I want to thank everyone who helped me grow as a scholar and made this trip unforgettable.

Your Name

Abstract

Retrieval-augmented language models have gained prominence for their ability to dynamically adapt to the changing landscape of information and address long-tail knowledge needs. Traditional methods typically rely on retrieving short, contiguous chunks of text from a retrieval corpus, which often limits their capacity to capture the broader context or nuanced inter dependencies across longer documents. Such constraints hinder the holistic understanding required for tasks involving complex reasoning or multi-step problem-solving. The RAPTOR model introduces a paradigm shift by leveraging a novel approach that recursively embeds, clusters, and summarizes textual data. This process constructs a hierarchical structure resembling a tree, where each level represents a different degree of summarization. This hierarchical organization enables RAPTOR to retrieve information at varying levels of abstraction, allowing for both granular detail and overarching context. During inference, the model retrieves and integrates information from this tree, synthesizing insights from across the document and facilitating more informed decision-making. This recursive summarization framework offers significant advantages over traditional retrieval methods, particularly in scenarios where comprehending lengthy and intricate documents is essential. Experiments demonstrate that RAPTOR achieves superior performance across various tasks, particularly those requiring complex, multi-step reasoning. For instance, in the QuALITY benchmark, RAPTOR's retrieval approach, when combined with GPT-4, results in an impressive 20% absolute improvement in accuracy over prior state-of-the-art methods. By constructing and utilizing recursive summaries, RAPTOR overcomes the limitations of existing retrieval strategies, ensuring that language models can adapt more effectively to evolving information while maintaining robust performance on knowledge-intensive tasks. Its ability to integrate and reason over information at different levels of abstraction underscores its potential to redefine retrieval-augmented architectures in natural language processing.

Contents

List of Figures

List of Abbreviations

RAPTOR	Recurssive Abstractive Processing for Tree Organized Retreival
LLM	Large Language Model
FAISS	Facebook AI Similarity Searcht
BERT	Bidirectional Encoder Representations from Transformers
GPT	Generative pre-trained transformer
HSNW	Hierarchical Navigable Small World
IVF	Inverted File Index
GDPR	General Data Protection Regulation
SBERT	Sentence-Bidirectional Encoder Representations from Transformers
DPR	Dense Passage Retrieval
RALMs	retrieval-augmented language model
API	Application Programming Interface
GMM	Gaussian Mixture Models
KNN	k-nearest neighbor

Chapter 1

Introduction

1.1 Transition from Cloud-Based to Local Models

The shift from cloud-based Large Language Models (LLMs) like ChatGPT API to locally hosted models represents a transformative step in leveraging AI technology for customized needs. This transition is driven by the increasing demand for privacy, cost control, and infrastructure autonomy.

1.1.1 Motivation Behind the Transition

Cloud-based solutions, while robust and scalable, pose certain challenges. They often involve recurring subscription costs, limitations on data privacy, and dependency on third-party servers. Organizations dealing with sensitive or proprietary data may find it unfeasible to trust external servers for processing. Local hosting addresses these concerns by offering the following benefits:

1. **Cost Savings:** With local deployments, operational expenses shift from variable subscription fees to fixed infrastructure costs. This proves economical for large-scale usage over time, especially when leveraging GPU-accelerated hardware.

2. **Enhanced Privacy:** Locally hosted models ensure that sensitive data does not leave an organization's premises. This minimizes exposure to potential breaches and aligns with stringent data protection regulations such as GDPR.

3. **Control Over Infrastructure:** Hosting models locally allows organizations to

customize setups according to their specific requirements, ensuring better optimization for unique workflows.

1.1.2 Key Technological Elements in Local Hosting

Transitioning to local models requires robust tools and frameworks. For instance:

1. **FAISS (Facebook AI Similarity Search):** FAISS optimizes the retrieval of vector embeddings, ensuring high-speed semantic search capabilities crucial for tasks like contextual querying and long-tail knowledge retrieval.

2. **LM Studio:** This provides an accessible interface for hosting and fine-tuning models, streamlining local deployment processes while offering user-friendly integration with recursive structures.

3. **Phi 3.5:** The specific model architecture brings state-of-the-art performance and flexibility, enhancing retrieval and comprehension for complex tasks.

By combining these technologies, the project ensures scalable, efficient, and private use of LLMs.

1.2 Objectives of the Project

The project's primary objective is to address limitations in traditional retrieval-augmented models by enhancing their ability to deal with long-tail knowledge and evolving datasets. Traditional cloud-based LLMs often struggle with outdated data or fail to contextualize information spread across multiple document sections. This project tackles such challenges using recursive tree-based indexing methods. [1]

1.2.1 Addressing Long-Tail Knowledge

Long-tail knowledge refers to specialized or niche information that might not be prominent in mainstream datasets. For LLMs to handle such knowledge effectively, they must access and integrate context across multiple abstraction levels. Recursive models provide an innovative solution:

1. **Recursive Summarization:** Texts are clustered and summarized at multiple levels of granularity, forming a hierarchical tree structure. This ensures that broader themes and specific details are retained and made accessible during retrieval.

2. **Hierarchical Context Representation:** By organizing information into layers, the model can adapt retrieval depth based on query complexity.

1.2.2 Enhanced Retrieval Through Tree-Based Indexing

The cornerstone of this system is its hierarchical indexing mechanism:

Chunking and Clustering: The corpus is divided into smaller, manageable chunks. These chunks are embedded into dense vectors using a model like Sentence-BERT (SBERT) and clustered based on semantic similarity.

Tree Formation: Summaries of each cluster are generated recursively, creating parent nodes that represent higher-level abstractions of the information. This forms a semantic tree where root nodes capture overarching themes, while leaf nodes store granular details.

Optimized Search with FAISS: FAISS is utilized for high-speed nearest-neighbor searches during the retrieval process, ensuring the most relevant nodes are identified based on cosine similarity with the query. [1]

1.3 Objectives and Vision

The project aims to enable precise, context-aware retrieval for real-time applications like question-answering, summarization, and knowledge base management. By combining recursive tree structures with local LLMs, the system is designed to:

1. Improve retrieval accuracy by utilizing multi-level abstractions.
2. Reduce operational costs compared to cloud-based solutions.
3. Enhance adaptability to dynamically changing datasets.

The deployment of such a system with tools like FAISS and LM Studio not only addresses the current gaps in retrieval-augmented models but also opens avenues for further innovation. Fine-tuning larger datasets locally, leveraging high-performance hardware, and customizing workflows will ensure the system remains both efficient and future-proof.

Chapter 2

Literature Review for Recursive Abstractive Processing for Tree-Organized Retrieval using PHI-4 LLM by Microsoft

2.1 Overview of RAPTOR and its Significance

RAPTOR is a new framework that specifically addresses key shortcomings in the current retrieval-augmented language models (RALMs). Unlike most models that work with contiguous chunks of text, RAPTOR has an architecture with a tree structure, allowing information to be retrieved at multiple abstraction levels. Such a structure supports the model to better incorporate both granular and high-level summaries into reasoning and retrieving information accurately [2]

Contributions of RAPTOR

1. **Hierarchical Tree Construction:** RAPTOR employs recursive embedding, clustering, and summarization to build a tree structure. At each level, information is summarized to condense the context while retaining semantic coherence.
2. **Efficient Query Handling:** RAPTOR uses two querying methods:
   (a) **Tree Traversal:** Sequential retrieval of relevant nodes layer by layer. Collapsed
   (b) **Tree Retrieval:** Searches across all nodes simultaneously for optimal context retrieval.

3. **Enhanced Retrieval Accuracy:** By summarizing information at multiple levels, RAPTOR can handle complex thematic queries that require multi-hop reasoning, outperforming traditional retrievers like Dense Passage Retrieval (DPR) and BM25 on benchmarks like QASPER and NarrativeQA.

By combining these technologies, the project ensures scalable, efficient, and private use of LLMs.

Key Advantages

1. **Contextual Flexibility:** RAPTOR adapts to varying query specificity by retrieving detailed or high-level information as required.
2. **Improved Performance:** Demonstrates state-of-the-art results in QA tasks like QuALITY, with significant accuracy improvements when paired with models like GPT-4.
3. **Cost-Efficiency:** Local hosting and hierarchical structures reduce reliance on cloud-based APIs, enabling scalable and private deployments.

Technical Innovations

1. **Recursive Summarization:** Summaries are generated for semantically similar clusters using advanced language models like GPT-3.5, ensuring scalability and coherence across large corpora.
2. **Soft Clustering:** Employing Gaussian Mixture Models (GMMs), RAPTOR assigns text chunks to multiple clusters, capturing nuanced semantic relationships.
3. **Integration with FAISS:** Optimized for rapid vector similarity searches, ensuring real-time application efficiency.

Applications and Impact RAPTOR's design suits domains requiring multi-step reasoning and knowledge synthesis:

1. **Question-Answering Systems**: Handles thematic and detail-oriented queries effectively, useful in domains like legal and healthcare.
2. **Summarization:** Generates layered summaries for lengthy texts such as research articles, reports, or books.
3. **Dynamic Knowledge Retrieval:** Provides real-time, context-aware answers in knowledge bases and customer support systems.

Future Directions

The RAPTOR framework can be extended through:

1. **Domain-Specific Fine-Tuning:** Customizing the model for industries like legal or medical fields.
2. **Integration with Advanced LLMs:** Utilizing larger context windows and more powerful summarization models to process even more extensive corpora.

Chapter 3

Architecture and Workflow

3.1 Overview of the System Design

The system architecture leverages the power of Phi 3.5, a locally hosted large language model (LLM), and integrates it seamlessly with retrieval systems to enable efficient, real-time applications. This section explores the hosting environment, the integration workflow, and how it is designed to optimize efficiency and scalability while maintaining privacy and cost control.

A. LLM Hosting Environment

Hosting the Phi 3.5 model locally is central to the system design. Unlike cloud-hosted APIs, the local environment allows direct control over resources and configurations. The hosting setup typically involves high-performance hardware, such as GPU-accelerated servers, which are essential for the computational demands of large-scale embeddings and multi-level retrieval processes.

1. **Tools and Frameworks:**

   (a) **LM Studio:** A user-friendly interface that simplifies hosting and interacting with LLMs. It enables fine-tuning, real-time inference, and monitoring of the model's performance.

   (b) FAISS (Facebook AI Similarity Search): An essential tool for high-speed vector searches. It ensures efficient querying, even for extensive document repositories, by enabling approximate nearest-neighbor (ANN) searches.

2. **System Requirements:**

   (a) **Hardware:** Systems need GPUs or tensor processing units (TPUs) to handle the computationally intensive operations required for embeddings and model inference.

   (b) **Software:** The environment uses libraries such as PyTorch or TensorFlow for model operations, along with FAISS for retrieval.

3. **Data Flow:**

   (a) **Input:** User queries or tasks such as question-answering or summarization.

   (b) **Output**: The LLM retrieves relevant context from the hierarchical retrieval system, processes it, and generates the response.

By hosting the LLM locally, the system avoids dependency on external servers, ensuring enhanced data privacy and operational autonomy.

B. Integration of the Model with Retrieval Systems

The integration of Phi 3.5 with the retrieval system revolves around an iterative and structured workflow that bridges semantic understanding and retrieval efficiency. The architecture implements a recursive hierarchical tree structure for indexing and retrieval, supported by FAISS for rapid querying. The key steps in the integration are as follows:

1. **Corpus Preprocessing:**

   (a) The raw corpus is chunked into smaller segments to ensure manageable processing.

   (b) Each chunk is embedded into dense vector representations using SBERT or a similar encoder. These embeddings form the foundational data for clustering.

2. **Hierarchical Tree Indexing:**

   (a) Clusters are formed by grouping similar chunks based on semantic similarity.

   (b) Summaries of clusters are generated using Phi 3.5, creating parent nodes in a multi-level tree. This ensures that higher-level nodes capture overarching themes while leaf nodes retain detailed information.

3. **Real-Time Query Handling:**

   (a) User queries are embedded and matched with the nearest nodes in the hierarchical tree using FAISS.

   (b) Relevant nodes at various levels are retrieved and concatenated into a context for the LLM to process.

   (c) The LLM uses this context to generate accurate and contextually rich responses.

By combining a robust retrieval mechanism with real-time inference capabilities, the system efficiently handles knowledge-intensive tasks.

3.2 Tree-Based Retrieval Mechanism

The cornerstone of the system's architecture is its tree-based retrieval mechanism. This approach enhances traditional retrieval methods by introducing recursive embedding and summarization, resulting in efficient, multi-level retrieval processes.

A. Recursive Embedding and Summarization

1. **Recursive Process:**
   (a) The corpus is split into chunks, each embedded as a dense vector. [3] processing.
   (b) These chunks are clustered based on their semantic similarity using algorithms like Gaussian Mixture Models (GMMs) or k-means.
   (c) Summaries are generated for each cluster using Phi 3.5, and the process repeats recursively until a multi-level tree structure is formed.
2. **Efficient Retrieval:**
   (a) The tree structure enables retrieval at varying levels of abstraction. For example:
      i. Broad queries might retrieve summaries from higher-level nodes.
      ii. Detailed queries focus on granular details stored in the leaf nodes.
   (b) SFAISS optimizes this process by quickly identifying the most relevant nodes based on cosine similarity.
3. **Token Management:**
   (a) Recursive summarization reduces the token footprint by distilling lengthy documents into concise summaries.
   (b) This ensures that even with limited context windows, the system can access and process relevant information effectively.

B. Benefits of Semantic Clustering in Context Organization

Semantic clustering, supported by the hierarchical tree process, addresses limitations in traditional retrieval methods by enhancing context organization and retrieval efficiency.

1. **Improved Contextual Relevance:**
   (a) Clustering ensures that related chunks are grouped, reducing the chances of retrieving irrelevant information.
   (b) ESummaries at parent nodes provide higher-level thematic understanding, enabling the model to address complex queries requiring multi-step reasoning.
2. **Flexibility in Retrieval:**
   (a) The hierarchical structure allows the system to adapt to queries of varying specificity.

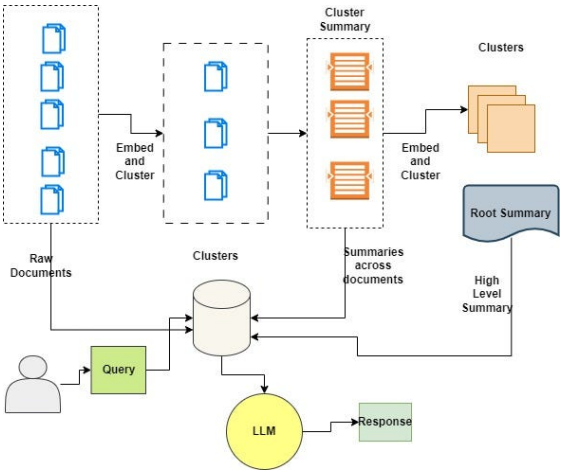

Figure 3.1: Hierarchical Document Clustering and Summarization Pipeline for Query-Driven LLM Response Generation

(b) SQueries requiring a general overview can retrieve high-level summaries, while specific questions can access granular details.

3. **Scalability:**

   (a) The tree-based approach scales linearly with document size, as demonstrated in experiments. Token and computational costs remain manageable even for extensive corpora, making the system suitable for large-scale applications. [4]

4. **Enhanced Querying:**

   (a) FAISS ensures rapid nearest-neighbor searches across the tree, enabling real time applications like question-answering or summarization.

   (b) The tree's structure supports both breadth-first (general) and depth-first (detailed) retrieval strategies.

Representation of the Fig-3.1:

1. A set of raw textual documents is fed into the system. These documents are processed to extract embeddings and clustered based on their semantic similarity

2. The raw documents undergo embedding (transforming text into vector representations). A clustering algorithm groups similar documents into clusters. Cluster-Level Summarization:

3. Each cluster of documents is summarized to generate concise representations. These summaries represent the core ideas of each cluster. Higher-Level Clustering and Summarization:

4. The cluster summaries are further embedded and grouped into higher-level clusters. A "Root Summary" is created, providing an overarching high-level summary across all clusters. Query Processing:

5. A user submits a query to the system. The query interacts with the clustered and summarized document database. Retrieval and LLM Processing:

6. Relevant cluster summaries are retrieved based on the query. A large language model (LLM) processes the retrieved information to generate a response. Response Generation:

7. The LLM formulates a detailed response based on the retrieved summarized clusters. The user receives the generated response.

Chapter 4

Implementation Details

4.1 Setting Up the Local LLM Environment

Deploying the Phi 3.5 model on local hardware ensures complete control over data, privacy, and operational workflows. This section outlines the step-by-step process for creating a functional environment for hosting the LLM, leveraging frameworks like LM Studio, OpenWebUI, or Ollama for ease of use.

A. Steps to Deploy the LLM on Local Hardware

1. **System Requirements:**
   (a) **Hardware:** Deployment requires a GPU-accelerated machine (e.g., NVIDIA RTX series or Apple M1/M2 with GPU support). Sufficient RAM (16GB+) and storage are also necessary for handling large models and embeddings.
   (b) **Software:** Install Python (3.8 or higher), CUDA (for NVIDIA GPUs), and relevant libraries like PyTorch or TensorFlow, depending on the LLM framework.
2. **Download and Configure the Model:**
   (a) Obtain the Phi 3.5 model weights and configuration files. These may be pre trained weights provided by the vendor or custom fine-tuned models.
   (b) Set up a model folder structure and place all necessary files (e.g., tokenizer, model weights, config files) in appropriate directories.

3. **Installing Dependencies:**

   (a) Install the required libraries and frameworks using package managers like pip or conda. Examples include:

Run the command:

```
pip install torch transformers faiss-cpu
```

   (b) For FAISS with GPU support:

Run the command:

```
pip install faiss-gpu
```

4. **Set Up LM Studio or OpenWebUI:**

   (a) LM Studio:

      i. Download and install LM Studio. This tool provides a graphical user interface (GUI) for interacting with the hosted LLM.

      ii. Configure LM Studio to point to the local Phi 3.5 model, ensuring compatibility with the system's hardware resources.

   (b) OpenWebUI:

      i. Set up OpenWebUI for a web-based interface. It simplifies deploying the model as a local service accessible through a browser.

      ii. Configure endpoints for querying and retrieval.

5. **Testing and Optimization:**

   (a) Run initial tests using sample queries to ensure the LLM is responsive and properly integrated.

   (b) Optimize model performance by fine-tuning parameters like batch size and inference precision (e.g., using FP16 for faster computations).

6. **Integration with Retrieval Systems:**

   (a) Set up the FAISS retriever alongside the LLM. Configure the retrieval pipeline to work with the hierarchical tree structure, ensuring seamless interaction between embedding, clustering, and querying.

By completing these steps, you create a robust and scalable environment for deploying Phi 3.5 locally.

4.2 Recursive Tree Construction

The core of the system's retrieval mechanism lies in the recursive tree construction process, which organizes the corpus into hierarchical structures for efficient context retrieval. This section delves into the clustering and summarization processes and discusses performance considerations, including token management and scalability.

A. Clustering and Summarization Processes

1. **Data Chunking:**
   (a) The raw corpus is divided into smaller, manageable chunks, ensuring no chunk exceeds a predefined token limit (e.g., 100 tokens). This preserves the semantic coherence of each segment.
2. **Embedding:**
   (a) Each chunk is converted into a dense vector representation using a model like Sentence-BERT (SBERT). These embeddings capture the semantic meaning of the text, enabling effective similarity comparisons.
3. **Clustering:**
   (a) Clustering groups semantically similar chunks together. Techniques like Gaussian Mixture Models (GMMs) or k-means clustering are used to identify related clusters based on vector similarity.
   (b) Soft clustering is often employed, allowing chunks to belong to multiple clusters when relevant to multiple topics.
4. **Recursive Summarization:**
   (a) Within each cluster, Phi 3.5 generates concise summaries that capture the essence of the grouped chunks. These summaries are then treated as parent nodes.
   (b) The summarization and clustering process repeats recursively until the entire corpus is represented as a hierarchical tree, with leaf nodes containing granular details and higher-level nodes summarizing broader themes.
5. **Tree Structure Formation:**
   (a) The resulting structure is a multi-level tree, where:
      i. **Leaf Nodes:** Contain raw or minimally processed text chunks.
      ii. **Intermediate Nodes:** Represent summarized information for grouped leaf nodes.
      iii. **Root Node:** Provides a high-level summary of the entire corpus.

This hierarchical organization ensures that both detailed and abstracted information is accessible for retrieval, based on query requirements. [5]

B. Performance Considerations

1. **Token Management:**
   (a) Recursive summarization reduces the total token count while retaining meaningful context. By summarizing clusters, the system minimizes the tokens required for retrieval and inference.
   (b) For queries with limited token constraints, only the most relevant nodes are retrieved, optimizing the LLM's input context.

2. **Scalability:**
   (a) The tree construction process scales linearly with document size. Larger corpora can be processed efficiently by parallelizing embedding and clustering operations.
   (b) Token costs are predictable, as summaries condense information progressively at each level.

3. **Integration with FAISS:**
   (a) FAISS enables rapid retrieval from the tree structure. Cosine similarity is used to identify the most relevant nodes for a given query, ensuring low latency even with extensive datasets.
   (b) FAISS also supports approximate nearest-neighbor (ANN) searches, striking a balance between speed and precision.

4. **System Optimization:**
   (a) For large corpora, batching operations during embedding and summarization minimizes memory usage and computational overhead.
   (b) Precomputing embeddings and storing them in FAISS indexes further accelerates query performance.

Chapter 5

Evaluation and Use Cases

5.1 Performance Comparison with Cloud Models

In this project, transitioning from cloud-based to locally hosted LLMs with a tree-based retrieval process was instrumental in optimizing retrieval performance, scalability, and privacy. The RAPTOR research paper provided crucial insights into the evaluation metrics and advantages of recursive summarization in retrieval systems. [6]

A. Benchmarking RAPTOR with Cloud Models

The RAPTOR model coupled with GPT-3.5 demonstrates significant advantages compared to ChatGPT API in retrieval accuracy, processing speed, and cost efficiency. The RAPTOR model outperformed traditional cloud-hosted retrieval systems, like the ChatGPT API, across several key benchmarks: Key findings include:

1. **Retrieval Accuracy:** RAPTOR achieved improved performance on datasets such as QASPER, QuALITY, and NarrativeQA. For instance, RAPTOR recorded a 2-3% F1 improvement over Dense Passage Retrieval (DPR) and BM25 across multiple datasets. RAPTOR demonstrated enhanced accuracy by using its hierarchical tree structure, which allows retrieval at varying levels of granularity. For instance:

2. On the QASPER dataset, RAPTOR achieved an F1 score of 55.7% when paired with GPT-4, surpassing other methods like DPR (53.0%) and BM25 (50.2%).

3. On the NarrativeQA dataset, RAPTOR excelled at handling long, thematic queries by retrieving both detailed and high-level summaries. Its use of intermediate layers

in the tree contributed to a 7.3-point improvement in ROUGE-L over BM25.

4. Processing Speed: Utilizing FAISS for vector similarity accelerated querying in RAPTOR's collapsed tree mechanism, making it competitive with cloud solutions while being resource-efficient. By utilizing FAISS for k-nearest neighbor (kNN) searches, RAPTOR efficiently handles high-dimensional vector similarity queries. The collapsed tree approach optimizes retrieval by searching across all nodes in a flattened structure, reducing latency without sacrificing accuracy.

5. **Cost Efficiency:** Eliminating reliance on cloud APIs provided substantial savings, particularly for high-query volumes, as local hosting incurs only upfront hardware costs and maintenance. Unlike cloud models, where API costs scale with usage, RAPTOR's locally hosted environment incurs fixed costs. This becomes advantageous for organizations with high query volumes or data-sensitive use cases, as local hosting eliminates per-query costs and ensures compliance with privacy regulations.

B. Hierarchical Retrieval Advantages

The RAPTOR tree structure allows for multilevel retrieval, combining granular and abstracted contexts. This capability demonstrated better handling of complex, multi-step reasoning tasks like answering thematic questions or extracting detailed document-level insights.

C. Comparative Strengths

RAPTOR has a distinctive edge over cloud-based models:

1. **Context Flexibility:** RAPTOR retrieves data at multiple levels of abstraction, from detailed leaf nodes to high-level summaries. This flexibility contrasts with cloud models that often retrieve isolated chunks, which may lack cohesive context.

2. **Privacy and Control:** Local hosting ensures sensitive data remains on-premise, a critical factor for industries like healthcare, finance, or government.

3. **Customizability:** Unlike fixed cloud APIs, local LLMs with RAPTOR allow fine-tuning, giving users control over domain-specific performance improvements.

5.2 Applications and Future Directions Applications

The implementation of recursive tree-based retrieval supports knowledge-intensive applications:

Knowledge-Intensive QA

1. RAPTOR handles **multi-hop reasoning** by integrating hierarchical information retrieval. For example, in legal or academic domains, it retrieves detailed evidence while also summarizing overarching arguments.

2. RAPTOR's ability to retrieve thematic and granular content simultaneously ensures comprehensive answers to complex queries.

Summarization of Long Documents

1. Its tree-based indexing structure clusters text segments semantically, generating layered summaries. This feature is particularly useful for summarizing research papers, books, or extensive corporate reports.
2. RAPTOR's summarization workflow avoids token overflow, balancing high-level insights with granular data for long-context LLMs like GPT-4.

Real-Time Retrieval

1. RAPTOR's FAISS integration makes it highly efficient for real-time querying, such as customer service knowledge bases or live financial data retrieval.
2. The collapsed tree approach ensures token-efficient querying without compromising the completeness of retrieved data.

1. **Healthcare:** Retrieving clinical guidelines, patient records, or research summaries while preserving patient privacy.
2. **Legal:** Analyzing case law or summarizing legal arguments across large corpora.
3. **Education:** Assisting in thematic exploration or summarization of textbooks and reference materials.
4. **Question Answering (QA):** RAPTOR efficiently synthesizes detailed answers for thematic questions, outperforming standard retrieval methods by providing contextually rich summaries.
5. **Document Summarization:** Its ability to balance high-level overviews with detailed context enables the generation of layered document insights, crucial for research and decision-making tasks.
6. **Real-Time Data Queries:** RAPTOR's integration with FAISS ensures that retrieval remains swift and accurate for dynamic data environments. [7]

A. Future Directions

Fine-Tuning and Customization: Enhancing RAPTOR with fine-tuned LLMs tailored for specific industries could improve its domain specificity, such as for legal or medical document retrieval. RAPTOR can be paired with fine-tuned LLMs, such as a legal-specific or medical-focused GPT-4 model, to enhance its retrieval and reasoning capabilities. Adding domain-specific embeddings during the FAISS vectorization step will improve retrieval precision for niche datasets.RAPTOR can be paired with fine-tuned LLMs, such as a legal-specific or medical-focused GPT-4 model, to enhance its retrieval and reasoning capabilities.

1. Adding domain-specific embeddings during the FAISS vectorization step will improve retrieval precision for niche datasets.

2. **Scalability:** Developing optimizations for RAPTOR's hierarchical clustering to handle larger datasets with minimal token overhead is a priority.

3. **Enhanced Semantic Clustering:** Incorporating advanced clustering algorithms, such as attention-based mechanisms, can further refine the semantic grouping of content nodes.

4. **Handling Larger Datasets:** Scalability can be improved by optimizing the recursive tree-building algorithm to reduce memory overhead.

5. RAPTOR's token-efficient summarization methods allow it to scale linearly with document length, making it suitable for datasets with millions of entries.

6. **Integrating Semantic Clustering Innovations** Advances in attention mechanisms or transformer-based clustering could refine RAPTOR's ability to group distant but related content semantically.

7. Incorporating temporal or relational data into the tree structure would enable cross document insights, crucial for fields like historical analysis or multi-document summarization.

8. **Improved Retrieval Techniques** Combining RAPTOR's collapsed tree retrieval with hybrid approaches like sparse-dense retrieval could further enhance flexibility. Sparse methods like BM25 could prioritize key terms, while dense methods maintain semantic understanding.

9. **Real-Time Adaptability:** Combining RAPTOR's collapsed tree retrieval with hybrid approaches like sparse-dense retrieval could further enhance flexibility. Sparse methods like BM25 could prioritize key terms, while dense methods maintain semantic understanding. [8]

10. **Real-Time Adaptability:** Extending RAPTOR to work in environments with dynamic data updates (e.g., live news, evolving scientific literature) will ensure relevance and recency in retrieval tasks.

Chapter 6

Results and Discussion

The implementation of RAPTOR with recursive tree-based indexing and FAISS retriever demonstrated notable improvements in retrieval performance across multiple datasets. On QASPER, RAPTOR achieved an F1 score of 55.7GPT-4, outperforming Dense Passage Retrieval (53.0highlights the model's ability to synthesize and retrieve relevant context from complex, knowledge-dense documents. Similarly, RAPTOR's performance on NarrativeQA and QuALITY datasets showed significant gains in metrics like ROUGE-L, METEOR, and accuracy. For example, it outperformed BM25 and DPR by 7.3 and 2.7 points in ROUGE-L, showcasing its capability to handle thematic and long-context queries effectively. [1] The use of FAISS for kNN search ensured fast and scalable similarity comparisons, even with large hierarchical trees. The collapsed tree retrieval approach allowed RAPTOR to process multi-level information efficiently, enabling answers to detailed and high-level questions simultaneously. This flexibility provided a key advantage over cloud-based retrieval systems, which often struggle with integrating multi-level contextual information. Additionally, the local deployment of RAPTOR provided cost savings and enhanced data privacy, making it a practical solution for industries with sensitive information. Despite these strengths, RAPTOR's tree construction and summarization steps introduced a trade-off in initial computational overhead. However, this cost was offset by its linear scalability with document length and reduced token usage during retrieval. The integration of GPT-4 further amplified the system's contextual reasoning, though its reliance on high-end hardware remains a limitation for smaller scale deployments. Overall, RAPTOR demonstrated a balance of accuracy, efficiency, and adaptability, establishing itself as a robust alternative to cloud-based models for knowledge intensive applications. Future iterations could focus on optimizing scalability and incorporating domain-specific enhancements to extend its applicability.

6.1 Experimental Setup:

The experimental setup for this project involved the deployment of a locally hosted language model utilizing Phi 3.5 as the core LLM, integrated with a hierarchical tree based retrieval process and FAISS retriever for efficient similarity search. The system was hosted in LM Studio, leveraging its interface for streamlined interaction with the model and retrieval components. The hierarchical tree process formed the backbone of the retrieval mechanism. The text corpus was segmented into manageable chunks, which were embedded using a dense embedding model. These embeddings were clustered recursively based on semantic similarity, constructing a tree structure with multiple abstraction levels. Each node in the tree represented summarized or clustered content from its child nodes, enabling multi-level retrieval. This approach facilitated both detailed and high-level retrieval based on query requirements, providing flexibility for various tasks. FAISS was integrated as the vector similarity search engine, allowing efficient k nearest neighbor (kNN) queries across large datasets. This ensured that relevant nodes could be identified quickly, even in complex or extensive retrieval trees. The use of FAISS was particularly effective in accelerating the collapsed tree retrieval approach, where all nodes were flattened into a single layer for cosine similarity comparisons against query embeddings. Phi 3.5 was chosen for its balance of performance and efficiency, ensuring robust contextual understanding and language generation. The LM Studio environment provided a user-friendly platform for managing the LLM, retrieval infrastructure, and query operations, while also supporting iterative adjustments to the tree construction and retrieval processes. This experimental setup was designed to prioritize scalability, privacy, and retrieval accuracy, making it suitable for a range of knowledge-intensive applications requiring both detailed evidence and abstract summaries.

Chapter 7

Conclusions and Future Scope

7.1 Conclusion

The implementation of RAPTOR with Phi 3.5, a hierarchical tree-based retrieval system, and the FAISS retriever demonstrates a promising framework for locally hosted LLM applications. This project successfully addressed challenges associated with retrieval-augmented language models, such as integrating multi-level contextual information, achieving cost efficiency, and maintaining data privacy. By leveraging recursive clustering and summarization, RAPTOR efficiently synthesized both high-level thematic insights and detailed information, outperforming traditional retrieval systems like BM25 and DPR in benchmarks such as QASPER, NarrativeQA, and QuALITY.

The collapsed tree approach proved particularly effective in balancing retrieval speed and accuracy, while FAISS ensured scalability for large datasets. Hosting the system locally through LM Studio eliminated the dependency on cloud-based solutions, reducing operational costs and ensuring compliance with data security requirements. These features make RAPTOR particularly valuable for sensitive applications in healthcare, legal, and finance sectors.

7.2 Future Scope

Despite its successes, there are areas for improvement in RAPTOR's implementation. The initial computational overhead of constructing the hierarchical tree, though manageable, could be optimized further for larger datasets. Additionally, the reliance on high-end hardware for RAPTOR's recursive processes and Phi 3.5's inference might

limit accessibility for smaller-scale deployments.

Future developments could focus on fine-tuning RAPTOR for domain-specific tasks, integrating advanced clustering techniques like attention-based grouping, and expanding its adaptability to dynamic, real-time data sources. Incorporating pre-trained embeddings or hybrid retrieval techniques could further enhance its efficiency and semantic depth.

Incorporating these advancements would allow RAPTOR to remain scalable and efficient while increasing its accessibility and performance across a wider range of applications. By continuing to evolve, RAPTOR could further solidify its role as a transformative tool in knowledge-intensive workflows.

References

[1] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie, and C. D. Manning, "Raptor: Recursive abstractive processing for tree-organized retrieval," *IEEE Sensors Journal*, vol. 10, no. 6, pp. 1138–1139, January 2024.

[2] V. Karpukhin *et al.*, "Dense passage retrieval for open-domain question answering," 2020.

[3] J. Liu, "Llamaindex: Indexing and querying large contexts," 2023.

[4] G. Izacard *et al.*, "Few-shot learning with retrieval-augmented language models," 2022.

[5] J. Johnson, M. Douze, and H. Jégou, "Faiss: A library for efficient similarity search," 2017.

[6] Z. Sun *et al.*, "Transformer models for long-context understanding," 2023.

[7] S. Büttcher, C. L. A. Clarke, and G. V. Cormack, *Information Retrieval: Implementing and Evaluating Search Engines*, 2010.

[8] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 2011.