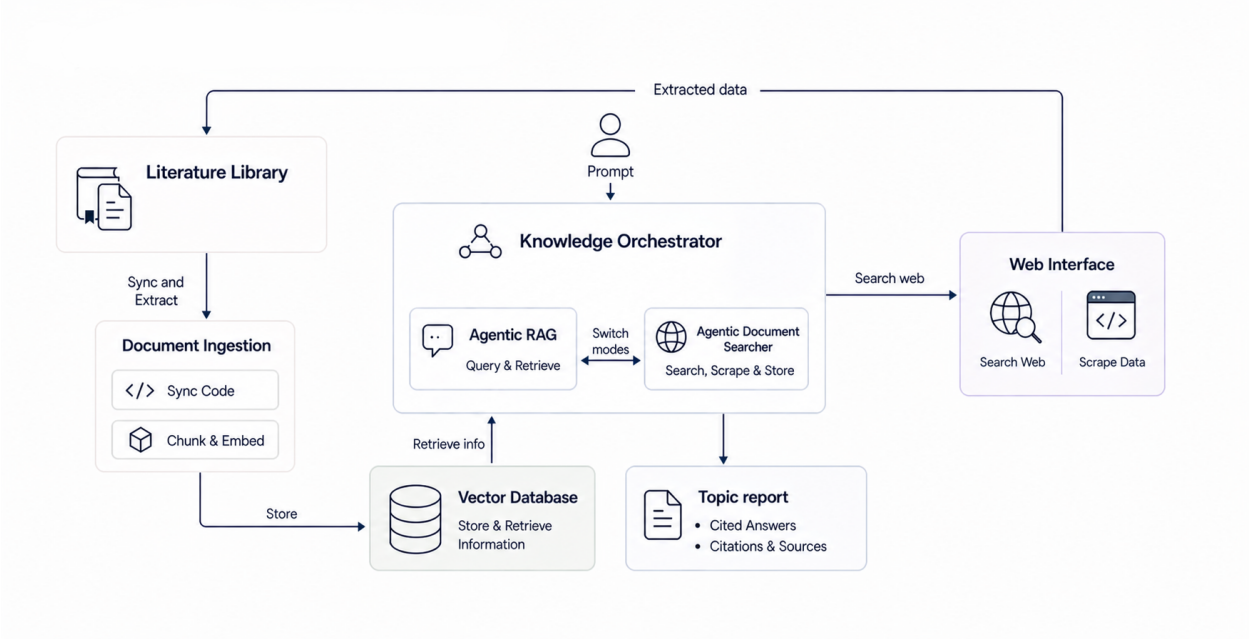
CHALMERS
UNIVERSITY OF TECHNOLOGY



Autonomous Knowledge Agent From Query to Report Without Human Intervention

Bachelor's thesis in Electrical Engineering

Noa Andreasen, Gustav Brochmann, Emil Günther, Gonzalo Rodriguez

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

BACHELOR'S THESIS 2026

Autonomous Knowledge Agent From Query to Report Without Human Intervention

Noa Andreasen, Gustav Brochmann, Emil Günther,
Gonzalo Rodriguez

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2026

Autonomous Knowledge Agent From Query to Report Without Human Intervention
Noa Andreasen, Gustav Brochmann, Emil Günther, Gonzalo Rodriguez

Supervisor: Carlos Natalino Da Silva & Kiarash Rezaei, Department of Electrical Engineering
Examiner: Paolo Monti, Department of Electrical Engineering

Bachelor's thesis 2026
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Cover: System architecture for the Autonomous Knowledge Agent, visualizing the flow of information and workflow. (Created with ChatGPT, 2026)

Typeset in LaTeX
Gothenburg, Sweden 2026

Autonomous Knowledge Agent From Query to Report Without Human Intervention
Noa Andreasen, Gustav Brochmann, Emil Günther, Gonzalo Rodriguez
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This thesis presents a locally executed autonomous literature search tool that treats the user's research idea as protected information. In this thesis, autonomous means that after the user provides a research query and validates the interpreted requirements, the system continues the literature search process without further human intervention.

The system combines four main components. The Agentic Document Searcher sources documents. The Document Ingestion system converts documents into searchable chunks . The Agentic RAG retrieves potentially relevant chunks and documents based on the information requirements derived from the user query. It uses an Information Foraging Theory-inspired classifier to distinguish between answering, interesting, and unrelated chunks, allowing the system to prioritize promising information patches and perform document deep dives when needed. The Knowledge Orchestrator coordinates the workflow from requirement interpretation to retrieval and final report generation.

The intuition behind the system is its design to run locally, ensuring stronger control over the user query when doing a literature search. The result is a working prototype that demonstrates how literature search can be automated while maintaining privacy for protected research input. At the same time, the evaluation indicates that performance remains limited by source coverage, relevance assessment, and document processing limitations.

Keywords: Agentic Document Searcher, Document Ingestion, chunks, Agentic RAG, Information Foraging Theory, Knowledge Orchestrator.

Acknowledgements

We would like to express our gratitude to our supervisors, Carlos Natalino Da Silva and Kiarash Rezaei, for their exceptional guidance and encouragement during this thesis. Their expertise in the field provided us with the necessary tools and insights to successfully design and evaluate our Autonomous Knowledge Agent. The frequent discussions and their constructive feedback helped us refine our ideas and resolve challenges. We would also like to express our gratitude to our examiner, Paolo Monti, for his valuable feedback and guidance throughout the thesis.

Noa Andreasen, Gustav Brochmann, Emil Günther, Gonzalo Rodriguez,
Gothenburg, May 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AI	Artificial Intelligence
API	Application Programming Interface
DOI	Digital Object Identifier
IFT	Information Foraging Theory
JSON	JavaScript Object Notation
LLM	Large Language Model
RAG	Retrieval-Augmented Generation

Nomenclature

Below is the nomenclature of parameters that have been used throughout this thesis.

Parameters

Min Coverage to Finish	Minimum requirement coverage score
Max Iterations	Maximum allowed iterations for the autonomous execution loop
Embedding Dimensions	Dimension of the vector embeddings
Embedding Batch Size	Batch size used for document embedding generation
Chunk Size	Chunk size in tokens
Chunk Overlap	Chunk overlap in tokens

Contents

List of Figures

List of Tables

1
Introduction

Literature search often begins with an uncertain research idea. Before developing the idea further, a researcher needs to understand how it relates to existing work. This requires several steps, including interpreting the research direction, finding relevant sources, evaluating their relevance, and synthesizing the findings into a coherent overview.

AI-based tools can support this process, but they also introduce a privacy problem. To receive useful assistance, the user may need to describe an unpublished, confidential, or commercially sensitive idea in detail. In such cases, the research query itself may contain information that should not be disclosed to an external service [6, 8].

This thesis presents a local-first autonomous literature search tool that treats the user's research idea and research query as protected information. In this thesis, autonomous means that after the user provides a research query and validates the interpreted requirements, the system continues the literature search process without further human intervention. The goal is to produce a readable, source-grounded topic report while preserving the privacy of the initial research query throughout the literature search process.

1.1 Problem and Motivation

The problem this thesis addresses is that a literature research query, that is the description used to describe the topic you want researched, is not always just a piece of non-confidential information. In early-stage research, the query can contain the actual research idea. To receive useful support from an AI-based literature search tool, the user may need to describe the idea in enough detail for the system to identify relevant concepts, search for related work, and judge whether the retrieved literature is relevant. If this description is sent to an external service, the user may reveal unpublished, confidential, or commercially sensitive information.

The risk is not mainly that the information is submitted once, but how it may be handled after submission. Depending on the service, the query may be retained, reviewed, shared with third parties, or used to improve machine-learning models [6, 8]. For sensitive research ideas, this can be problematic because the value of the idea may depend on keeping it confidential until it has been protected, published,

or properly attributed.

For instance, a researcher developing a new hypothesis about a previously untested biomarker for early cancer detection may need to describe the idea in order to understand whether it has already been studied and how it relates to prior work. In doing so, the researcher may reveal the core of an unpublished research direction before it has been protected, published, or properly attributed.

In this thesis, protected information refers to natural-language research input that should satisfy the following guarantees:

1. it is not used for training AI models;
2. it is not shared with third parties;
3. it is not retained beyond what is strictly necessary for the task;
4. it is not accessible beyond the components required to generate the response.

This motivates a local-first execution setting, where the user keeps stronger control over how the research query is processed. This does not mean that externally hosted services are inherently unsuitable for research use, since different providers offer different levels of privacy. However, for the use case addressed in this thesis, the user needs explicit guarantees about how protected information is processed, stored, accessed, and reused.

1.2 Contributions

This thesis contributes a working local-first autonomous knowledge agent designed for privacy-preserving literature search. Rather than presenting a generic retrieval-augmented generation (RAG) pipeline, this system introduces a specialized architecture where privacy, relevance assessment, and evidence sufficiency are treated as core design concerns.

The specific contributions of this work are as follows:

1. **A Local-First Architecture for Autonomous Literature Search:** An architecture is defined that integrates query interpretation, academic source search, document ingestion, retrieval, relevance assessment, and report generation into a unified autonomous local workflow. By keeping model inference and data storage on local hardware, the architecture ensures that the user's research idea and research query are treated as protected information and remain under the user's control.
2. **An Iterative Agentic Query-to-Report Workflow:** The system is structured as an iterative and cyclic agentic workflow rather than a single retrieve-and-generate pipeline. This design allows the system to autonomously asses the amount of sufficient data, determining whether to continue the search for additional sources, or to finalize the report based on the degree to which

gathered information satisfies the user's query.

3. **A Chunk-Level Relevance Classifier for Agentic Retrieval:** The system introduces a classification step between retrieval and report generation. Retrieved chunks are classified as answering, interesting but insufficient, or unrelated. These categories allow the Agentic RAG to keep direct evidence, discard unrelated material, and use partially relevant chunks as signals for further document exploration. The design is inspired by Information Foraging Theory, where retrieved chunks are treated as information patches and partially relevant chunks provide information scent.
4. **A Grey-Box Feasibility Evaluation of the Autonomous Workflow:** The system is evaluated across three metrics: scope, correctness, and usefulness. The evaluation considers both the generated topic reports and selected internal pipeline outputs, making it possible to connect report-level weaknesses to specific system limitations such as source coverage, ingestion reliability, relevance classification, and evidence depth.

1.3 Report Structure

The remainder of this report is organized as follows.

- Section 2 presents the background needed to understand the project, including the relevant technologies, the research domain, and existing AI-based research tools.
- Section 3 describes the method, including the system architecture, implementation environment, technical stack, subsystems, and system configuration.
- Section 4 presents the feasibility study, including the evaluation results, weaknesses, improvements, ethical aspects, and future work.
- Section 5 presents the final conclusions and synthesizes the main takeaways of the thesis. The report ends with a bibliography and appendices.

2
Background

This section provides the context needed to understand the thesis. It introduces the main technologies behind autonomous research agents, explains how these technologies are used in research settings, and compares existing AI-based research tools in relation to the focus of this work.

2.1 Technology

The technological background of this thesis is based on large language models (LLMs), retrieval-augmented generation (RAG), document retrieval, and agentic workflows. LLMs make it possible to interpret natural-language instructions, generate coherent written output, and perform increasingly complex reasoning tasks [16]. Longer context windows have also improved the ability of LLMs to process larger amounts of text within a single interaction [3]. These capabilities are important for building systems aimed to automate literature search, where the system must read and write vast amounts of scientific text.

However, LLMs alone are not sufficient for research-oriented tasks. If an LLM generates an answer only from its internal knowledge, the result may be difficult to verify or may not reflect the specific literature relevant to the user's query. RAG addresses this limitation by allowing a model to use retrieved external material during generation [4]. This is important in literature search because the generated report should be grounded in identifiable sources rather than unsupported model output.

A key requirement for RAG-based literature search is that documents must be represented in a form that can be searched effectively. Scientific papers are usually too long to be used as whole documents in a single model prompt, so they are commonly divided into smaller passages or chunks. These chunks can then be retrieved through some sort of lexical search, which matches explicit terms, or through semantic search, which uses vector representations to find passages with similar meaning [4, 15]. This combination is useful in research settings because relevant studies may use different terminology while still discussing similar concepts.

Agentic workflows extend this further by structuring complex tasks into multiple stages, such as planning, retrieval, evaluation, and synthesis [1, 2]. This is relevant because literature search is naturally iterative: the system may need to search for sources, judge whether the available evidence is sufficient, and continue searching

when important information is missing. Together, these technologies provide the basis for a system in which LLMs interpret and synthesize information, retrieval methods ground the output in source material, and agentic workflows decide when further search is needed. This technological basis also supports the privacy goal introduced in Section 1, since the main processing steps can be executed in a local-first setting.

2.2 Research Domain

This thesis is situated in the domain of AI-supported research workflows, where LLMs are increasingly used to help researchers search for, organize, summarize, and write about scientific information. In 2023, when LLMs began receiving widespread attention in both public and research settings, their role in research was primarily framed as support for individual subtasks, including information retrieval, knowledge organization, and writing, often through proprietary chatbot-based services such as ChatGPT and related systems [5]. In these settings, the model typically acts as an assistant that responds to narrowly scoped prompts rather than as a system that independently carries out a broader research process.

At the same time, concerns regarding privacy, security, and data governance became central [6]. The European Commission's guidelines on the responsible use of generative AI in research emphasize that researchers should pay attention to privacy, confidentiality, intellectual property rights, where a tool runs, who manages it, and how submitted information is handled [8]. These concerns are especially relevant when the information submitted to an AI tool contains unpublished research ideas or other sensitive material.

Reliability is another important concern in research settings. For LLM-based systems to produce domain-relevant outputs with an acceptable level of accuracy, they must be grounded in domain-specific material [4]. On their own, LLMs are limited to the knowledge encoded during training, together with the context provided in the prompt. This is often insufficient when outputs need to be traceable, source-grounded, and possible to verify.

More recently, deep research agents have begun to extend LLM usage beyond simple assistance by autonomously carrying out multi-step research tasks grounded in external sources [2]. This illustrates the potential for automating larger parts of the research workflow. However, access to academic material remains a practical limitation for research tools, especially when relevant sources are behind publisher restrictions or are not available through the tool's connected sources [12].

2.3 Existing AI-based Research Tools

Several AI-based tools already support researchers with tasks such as finding papers, reading documents, summarizing content, and organizing literature. However, these tools are not all designed for the same purpose, and they therefore do not address the same problem as this thesis. Table 2.1 compares selected tools in terms of their scope of support, stated privacy position, and the gap relative to this thesis.

The comparison separates two types of gaps. First, if a tool does not provide literature search, it is considered outside the main use case addressed in this thesis, and its privacy policy is therefore not analyzed further in relation to protected research queries. Second, if a tool does provide literature search, the relevant gap is whether it explicitly states the privacy guarantees required for protected information in this thesis, such as whether user queries are retained, shared with third parties, or used to train AI models. In this way, the table does not claim that the compared tools are unsuitable in general, but clarifies whether they combine literature search with the explicit privacy guarantees required for the protected-information setting addressed here.

Table 2.1: Comparison of related AI-based research tools

Tool	**Scope of support**	**Privacy stance**	**Gap relative to this thesis**
Avidnote	Writing, reading, note-taking, and single document interaction [7].	States that user data remains private and is not used to train their AI model [7].	Does not offer literature search.
ResearchRabbit	Citation-based literature discovery and paper recommendations [9].	States that user's articles and notes are not used to train AI models and are not sold or shared with third parties [14].	Does not offer literature search
SciSpace	Literature search, paper analysis, summarization, comparison, and drafting [10].	States that uploaded PDFs are private and are not used to train its AI models [11].	Does offer literature search but does not explicitly state protecting user queries.

Together, these tools show that AI-based research support already covers several parts of the research workflow, but they do not fully address the focus of this thesis: Autonomous literature search where the initial research query is handled as protected information.

SciSpace is the most closely related tool because it explicitly supports literature search. However, its cited privacy guarantee concerns uploaded PDFs rather than

the query-level assurances some researchers may need before entering a sensitive research idea, such as training use, third-party sharing, retention, and access control [11].

3
Methods

This section describes the design and implementation of the autonomous knowledge agent, which transforms a research query into a source-based topic report. The system is built around four primary components: the Knowledge Orchestrator, Agentic Document Searcher, Document Ingestion, and Agentic RAG.

To address the core project constraints of data privacy and local control, the methodology follows a local-first logic where all data processing and model inference are isolated from external cloud services. The following subsections first explain the system architecture to establish the workflow, followed by the technical stack, where specific design choices, such as the use of local vector databases and local LLMs, are justified against these privacy requirements. Finally, the functions of each subsystem are explained to demonstrate how they collaborate to form the complete system.

3.1 Architecture

The system is composed of three primary subsystems orchestrated by the Knowledge Orchestrator, the central unit. Acting as the primary decision-making agent, the Orchestrator manages the workflow by delegating tasks to specialized subsystems to fulfill user requests. A visual layout of this architecture is provided in Figure 3.1, illustrating the flow of data from the initial user prompt to final report generation.

The process begins when the Knowledge Orchestrator receives a user prompt and generates a set of interpreted requirements, used by the orchestrator as a metric for when enough information has been collected. These are returned to the user for validation, allowing for manual edits or approval to ensure the requirements align with the user's intent.

Once the requirements are confirmed by the user, the Orchestrator invokes the Agentic RAG to query the local vector database. If the retrieved information is deemed insufficient based on predefined relevance thresholds, the Agentic Document Searcher is activated. This subsystem performs targeted searches across academic databases. Found documents are integrated into a literature library, after which they are chunked and embedded by the Document Ingestion for database ingestion. This iterative loop continues until a sufficient knowledge base is established to satisfy the research query. Finally, the Knowledge Orchestrator synthesizes the gathered data

into a structured, source-grounded topic report.

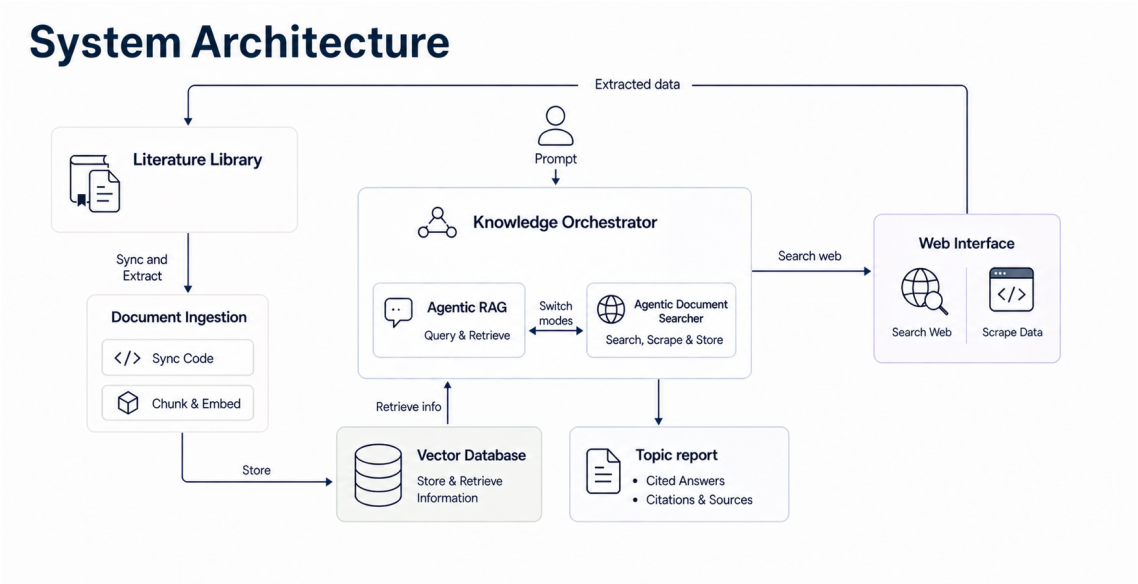


Figure 3.1: System architecture visualization (Created with ChatGPT, 2026)

3.2 Implementation Environment and Technical Stack

This subsection presents and justifies the technical selections made to satisfy the system's architectural requirements. The core stack was chosen to prioritize local-first environments, ensuring that no sensitive data is exposed to any service that doesn't fulfill our requirements for privacy.

3.2.1 Core Orchestration and Logic

The system is implemented in Python, which was selected over alternative languages like JavaScript due to Python's vast ecosystem of AI libraries, such as Ollama and LangGraph. While lower-level languages might offer higher raw execution speed, the primary performance bottlenecks in the system are LLM inference and document parsing rather than raw code execution. LangGraph was specifically chosen over the standard LangChain "chains" since the system workflow is iterative and non-linear. Unlike standard sequential chains, LangGraph supports stateful, cyclic workflows, allowing the system to manage multi-step reasoning, tool orchestration, and branching decisions across the workflow. To manage asynchronous communication between the subsystems and the interface, FastAPI and Uvicorn were selected for their high performance and efficiency.

3.2.2 Local Inference and Storage

To satisfy the project's strict privacy requirements, the system is designed around a local-first logic. Under this logic, all computations and most storage solutions

are hosted locally. Ollama was chosen for local LLM deployment, providing a privacy-secure alternative to cloud hosted LLMs such as those provided by OpenAI. While cloud based models usually offer longer context windows and greater reasoning, Ollama ensures that all data remain local and never leave the user hardware. Llama3:8b was selected as the primary model for inference, alternatives like qwen3:30b have longer context windows, but slower execution speeds. Hence, llama3:8b provided the most effective balance between processing speed and context window length.

Marker is used to convert PDFs into structured Markdown, a critical step for effective document ingestion. Unlike standard extraction tools, Marker preserves the logical hierarchy of complex scientific layouts. Furthermore, Marker runs locally, satisfying the project's local-first logic. While PDFMiner could serve as an alternative, it is less reliable than Marker and carries the risk of failing to correctly recognize columns and tables.

PostgreSQL with the pgvector extension is used as the local retrieval database. Documents are divided into chunks, converted into vector embeddings, and stored in the database to support semantic similarity search. Alternatively, cloud based databases like Pinecone could have been used instead, although this would not meet the privacy constraints.

The only exception to the local-only storage is Zotero, which is used for literature management. Zotero was selected over building a custom local file management system because it provides a researcher-friendly API for managing PDF attachments and metadata.

Finally, the user interface is implemented with Next.js, React, and TypeScript. This stack was chosen over simpler dashboard tools like Streamlit to provide a more robust, production-ready environment capable of tracking and displaying the complex state transitions of the Knowledge Orchestrator in real-time. All system settings, including LLM provider, database connection, Zotero access, embedding configuration, document parsing options, and runtime limits, are managed through a centralized environment configuration file to ensure the system is portable and easily configurable for different local hardware setups.

3.3 Subsystems

This subsection describes the design and function of the four subsystems, as well as how they work together to form the complete system.

3.3.1 Knowledge Orchestrator

The Knowledge Orchestrator serves as the central control unit of the system and is responsible for coordinating the workflow between the various subsystems. The

Knowledge Orchestrator is implemented using the LangGraph framework, which allows for a StateGraph where the logic is represented as a directed graph of nodes. At the core of this architecture is the concept of a shared state object that is passed between nodes, containing all the critical information required for the system to function autonomously. This state includes the requirements, evidence extracted by the Agentic RAG, a coverage score indicating how much of the initial user query has been satisfied, and a counter to track the number of iterations performed.

The process begins with the orchestrator generating a set of requirements based on the user's query. This step transforms a potentially vague prompt into a set of concrete, measurable requirements that together with the user's query serve as the system's objective. Once these requirements are validated by the user, the orchestrator enters an autonomous execution loop. For each iteration, the orchestrator evaluates the current state to decide the next step. It decides whether the system should proceed to the Agentic Document Searcher to acquire more information, activate the Agentic RAG for deeper analysis of the current information in the vector database, or finalize the process by writing the final topic report.

A vital function of the Knowledge Orchestrator is its role as a decision-maker regarding current information sufficiency. By analyzing how well the current available information compares to the requirements and user query, the Knowledge Orchestrator calculates a coverage score. If the score meets the threshold of 0.85, or if the maximum number of allowed iterations is reached, the Knowledge Orchestrator stops the iteration of the system. This design ensures that the system remains focused on the user's goal, while preventing the system from running an infinite number of loops, which wastes resources and increases the risk of the LLM starting to hallucinate. By managing the high-level logic, the Knowledge Orchestrator enables the system to operate as a fully autonomous agent that maintains context and direction throughout the entire process.

3.3.2 Agentic Document Searcher

The Agentic Document Searcher works as a search tool that converts the user query and requirements into a database-specific API query and returns relevant academic sources for later processing. The current implementation uses OpenAlex and arXiv, where the two sources support slightly different filters, hence the function of each source differs slightly.

The Agentic Document Searcher begins its work when it receives the user query and the requirements from the Knowledge Orchestrator. The first step is to extract keywords and constraints from the input, where the keywords are used for the actual API query, and the constraints are used as filters for the search.

An LLM extracts the most significant keywords from the input, prioritizing technical terms, domain-specific identifiers, and content-bearing nouns to optimize the search. For example, in the query "Autonomous navigation system for UAV using

AI-vision models" the system identifies "Autonomous", "navigation", "UAV", "AI-vision" as high-value keywords.

Constraints are prepared by the Knowledge Orchestrator before the document searcher is called. For example, if the user asks for papers “within the last 5 years”, this is stored as a recency constraint and can be converted into a publication-year filter in the OpenAlex query. Apart from the constraints prepared by the Knowledge Orchestrator, the document searcher also applies predefined database-specific filters, such as OpenAlex filters for open access, abstracts, and PDF availability.

When the system has extracted keywords and constraints, an API query is built using the keywords that is used to search the two academic databases for relevant articles. The search is configured to be restrictive, requiring all keywords to exist within either the title or the abstract of the article. While OpenAlex supports the ability to filter results by specific publication years, arXiv does not, requiring the system to handle those constraints differently.

The found articles are deemed as candidates; the next step is for an LLM to evaluate the found articles based on the user query and the requirements given from the Knowledge Orchestrator. The abstract and the title from the candidate articles are downloaded and evaluated by the LLM. The LLM scores candidate papers before they continue in the pipeline. In the arXiv path, the prompt asks for a 0–10 relevance score, and papers are accepted when the parsed score is at least 4. In the OpenAlex path, the LLM evaluates candidate papers on a 1–5 scale, where a score of at least 3 is required for the paper to be accepted.

For articles that pass the evaluation, the full PDF is downloaded and then uploaded to the literature management system Zotero, where the articles are traced and processed by the other subsystems. The uploaded data includes the full article and metadata such as publication year, authors, file size, DOI, and the source database together with the source URL.

3.3.3 Document Ingestion

The Document Ingestion is the subsystem responsible for retrieving, processing, and storing research papers, preparing them for the downstream RAG search agents. Because scientific documents are complex, often containing multi-column layouts, tables and mathematical formulas, they must be pre-processed so the Agentic RAG can accurately retrieve required information. The ingestion pipeline acts as a fully automated extract, transform and load workflow, sequentially moving documents from an initial reference manager to a vectorized database schema designed to support both semantic and lexical searches. The pipeline intelligently skips previously processed documents unless their underlying file or metadata has changed.

3.3.3.0.1 Data Acquisition The ingestion process begins by orchestrating a connection to Zotero, which serves as the primary literature repository. A dedicated

connector authenticates with the Zotero API and streams the metadata for the active collection of documents, along with their associated PDF attachments. To minimize computational cost, the service checks the exact file hash of each PDF attachment. If an identical document has already been processed, it is not ingested again. If a new document is detected, the associated PDF is downloaded for further processing.

3.3.3.0.2 Document Parsing A major challenge in RAG applications is accurately extracting structured text from PDFs for retrieval at the retrieval step. To handle this, the downloaded papers are processed by a parsing service powered by the Marker engine. The parsing step takes the raw PDF and transforms it into a highly structured, machine-readable Markdown format. Unlike naive text extraction tools, Marker actively preserves the inherent logical structure of the paper, converting hierarchical section headings, paragraph breaks and even page boundaries into Markdown elements.

3.3.3.0.3 Chunking Strategy Once the structured Markdown representation is obtained, the text must be partitioned into segments small enough to fit within an LLM's context window, yet large enough to contain actionable meaning. The chunking module employs a structural strategy by first splitting the Markdown into sections, bounded by its headings. It tracks the hierarchical depth of each section, accumulating a full descriptive path (e.g., *Method > Data Collection > Preprocessing*). Within these sections, the text is further divided into sliding token windows. Each produced text chunk independently retains metadata regarding its parent document, character offsets, token counts, and its corresponding section path context.

3.3.3.0.4 Embedding and Storage The final stage of ingestion involves indexing the generated chunks so they can be effectively retrieved. Each chunk's raw text is passed to an embedding model to generate a dense numerical vector representation capturing its semantic meaning. Finally, the original document metadata, the full Markdown output, and the sequence of individual chunks paired with their vector embeddings are stored within a PostgreSQL database optimized with the `pgvector` extension.

3.3.4 Agentic RAG

The Agentic RAG is the subsystem responsible for retrieving and evaluating the papers that are used in the topic report. It does this through predefined workflows, with agentic decision-making incorporated when the system needs to decide how to continue the search. Moving the literature search process from a human researcher to an agent follows a similar approach. In both cases, the process consists of repeatedly searching, reading, evaluating, and deciding whether a piece of literature is relevant enough to continue with. If the literature does not align with the target topic, the researcher or agent moves on. If it does align, more time is spent examining it. This

is why the Agentic RAG workflow is based on principles from Information Foraging Theory (IFT) [13]. Since the theory describes how information seekers search, filter, and decide whether to continue exploring a source, it provides a useful basis for designing an effective retrieval workflow. In this system, the theory motivates design decisions such as using relevance signals to asses retrieved chunks, discarding unrelated information early, and expanding the search only when a source appears promising.

3.3.4.0.1 Information Foraging Theory IFT describes how humans try to maximize the rate of gaining valuable information while minimizing the effort required to find it [13]. This makes it relevant for the design of an agentic retrieval system, since the agent faces a similar problem: it must decide where to search, which results to follow, and when enough relevant information has been found, with respect to time and computation spent.

In IFT, information patches can be understood as places where information is located, such as papers, sections or abstracts. The theory assumes that information seekers often move between patches and must decide how much time to spend within a current patch before leaving it to search for another one. This creates a trade-off between exploiting a current patch and searching for a potentially better one. Information Foraging Theory is therefore concerned with how information seekers allocate effort between within-patch activities, where they examine a current patch more deeply, and between-patch activities, where they move to other sources. The RAG system must navigate vast amounts of information patches, and it can't process them all due to time and computation limits.

Information scent refers to the cues that indicate whether a patch is likely to contain useful information, such as paper titles, keywords or section headings. The stronger the scent, the more reason the agent has to continue processing that patch. If the scent is weak, the agent should move on and search elsewhere. This principle is useful for our RAG solution because it supports a search process where the agent does not treat all retrieved papers equally, but instead prioritizes the sources that seem most likely to contribute valuable information to the final topic report.

Information scent refers to the cues that indicate whether a patch is likely to contain useful information, such as paper titles, abstracts, keywords or section headings. Since the full value of an information source is usually not known before it has been examined, information seekers rely on these cues to estimate whether a source is worth pursuing. In this sense, scent functions as an early relevance signal. It helps the forager decide which paths, sources, or patches are likely to provide valuable information relative to the cost of processing them. Information scent is important because it allows the forager to filter and prioritize information before committing to deeper processing. A source with strong scent gives the forager a reason to continue exploring it, while a source with weak scent may be skipped in favor of another source with a higher expected value. This means that information seeking is not only a process of finding sources, but also a process of continuously evaluating whether

the available cues justify further attention. Filtering based on scent therefore reduces the risk of spending time on low-value information and increases the chance that effort is directed toward more profitable patches.

3.3.4.0.2 RAG workflow A RAG task's input is the user query and the requirements derived from the query. The system will attempt to retrieve the answer to each requirement by following the abstraction in Figure 3.2. This will hence be ran as many times as the amount of requirements provided. The orchestration of the workflow is built with LangGraph's StateGraph. The graph consists of a starting node, summary node, deep dive node and end node. Each node accumulates information relevant to the requirements and adds to the state, except for the end node which reads the state and builds a coverage result for the Knowledge Orchestrator, containing the information retrieved and how well the requirements was answered.

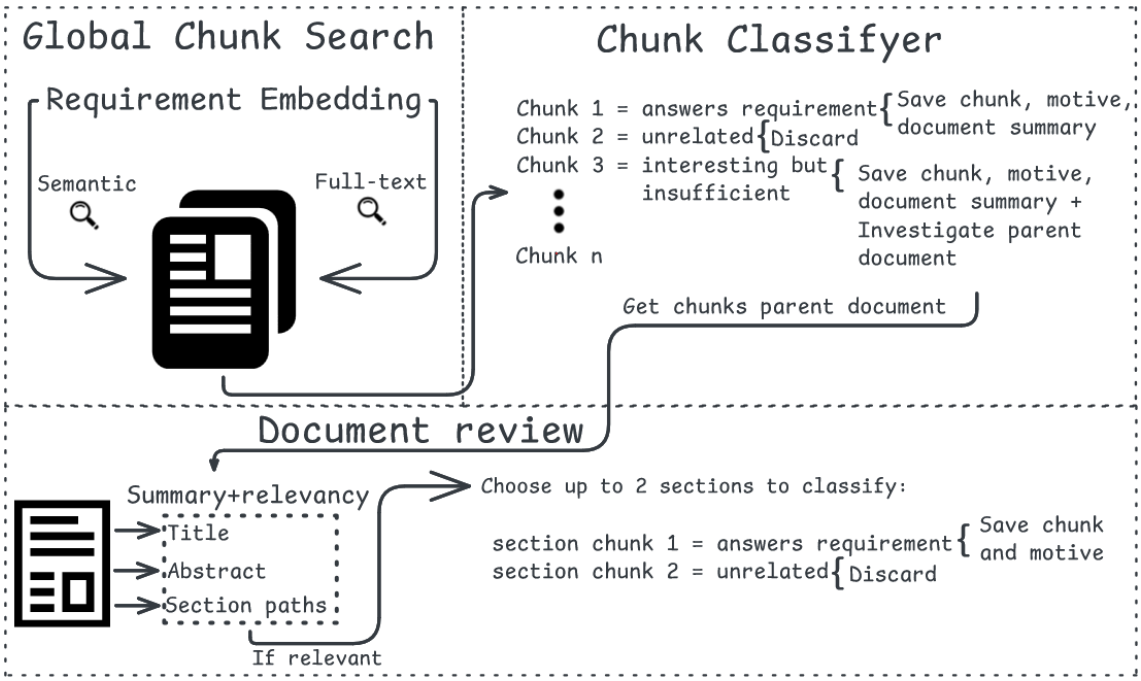


Figure 3.2: RAG architecture

3.3.4.0.3 Starting Node The starting node retrieves the initial chunks through both semantic search and full-text search. These chunks are treated as initial information patches, meaning initial places in the corpus where useful information may be found. Full-text search provides this starting point by following explicit information scent, such as keywords and terms that match the requirement. Semantic search provides a broader form of scent by finding chunks whose meaning or sentence-level content overlaps with the topic being investigated, even when the exact same words are not used. Together, these searches serve the agent with chunks that function as information patches likely to contain useful evidence for answering the requirement.

The full-text search is performed by converting each requirement into a PostgreSQL full-text search query. First, the requirement is processed so that the search focuses on content-bearing terms rather than common stop words. PostgreSQL's `websearch_to_tsquery()` function is then used to convert the requirement into a `tsquery` object, which represents the keywords and search logic used in the database query. The text stored in each chunk is converted into a searchable token representation using PostgreSQL's `to_tsvector()` function. During this process, common stop words are removed and words are normalized to their stems. This means that different grammatical forms of the same word can be matched against the same underlying term. For example, words such as "search", "searching", and "searched" can be treated as related forms. The search is then performed using PostgreSQL's `@@` operator, which compares the `tsquery` object from the requirement with the `tsvector` representation of each chunk. If the query terms match the chunk representation, the chunk is returned as a candidate result.

Semantic search provides a broader form of information scent by retrieving chunks whose meaning is similar to the requirement, even when the exact same keywords are not used. The requirement is converted into an embedding vector using the same embedding model that was used when the chunks were ingested. This vector represents the semantic content of the requirement in a numerical form. The chunks in the database have already been embedded during the document ingestion process and stored as vectors in PostgreSQL using pgvector. These vectors are indexed with an HNSW index, which makes it possible to efficiently search for chunks that are close to the requirement vector in the embedding space. In this context, closeness means that the requirement and the chunk are semantically similar according to the embedding model. The semantic search is then performed by comparing the requirement embedding against the stored chunk embeddings. Chunks with the smallest vector distance, are returned as candidate results.

Together, the full-text search and semantic search provide two different forms of initial information. The system is configured to return three candidates of each to limit time and computation spent. The retrieved chunks then function as candidate information patches that may contain useful evidence for answering the requirement, or serve as information scent for further investigation.

The retrieved chunks are then classified by an LLM into three categories: answering, interesting but insufficient, or unrelated. The model used for this step is llama3:8b. The classifier is prompted to act as a strict retrieval judge and to return a structured JSON response. The prompt used for the classification is shown below:

```
You are a strict retrieval judge.
Judge how well the chunk answers the requirement derived from the query.
Return JSON only with keys: tag, motive.
Allowed tags: answers_req, interesting but not enough, unrelated.
Input JSON: {
  "query": "<user query>",
```

```
  "requirement_text": "<current requirement>",
  "chunk_text": "<retrieved chunk text>"

}
```

This classification acts as a filtering step based on the information scent found in each chunk. An unrelated chunk is treated as having weak or irrelevant scent and is therefore discarded. An answering chunk is treated as containing enough information value to answer the requirement and is saved together with the motive for why it answers. An interesting but insufficient chunk is also saved together with the motive, however it occupies an intermediate position: it does not contain enough information to answer the requirement directly, but it contains enough scent to suggest that it relates.

This is where the workflow differs from a traditional RAG approach. In a simpler RAG pipeline, retrieval is often followed directly by generation, meaning that the system attempts to answer the query using only the initially retrieved chunks. In this workflow, IFT motivates an additional decision step in which interesting chunks are treated as information scent towards the parent document which is covered in the Deep Dive Node.

3.3.4.0.4 Summary Node Both interesting and answering chunks require document-level summarization since it will be provided to the Knowledge Orchestrator as supporting context for the topic report. This gives the system a broader understanding of the document in which the answering chunk appeared. For interesting chunks the summary will also be used in the Deep Dive Node as information scent.

The summary node summarizes the parent document of retrieved chunks. The model used for this step is llama3:8b. The purpose is not to summarize the full document, but to create a short document-level overview from the same types of information scent that a human reader would typically use to judge what a paper is about: the title, the abstract, and the section paths. If no abstract is available, the introduction chunk is used instead (if no introduction section is available, it settles without it). A section path refers to the hierarchical heading structure of a section, such as Methods > Data Collection > Preprocessing. For the summary step, the section paths of all chunks in the document are provided. This gives the model an overview of the document's structure without passing the full document text. The prompt used for the summary step is shown below:

```
You are a concise scientific document summarizer.
Use title + abstract/introduction context only.
Return JSON only with key: summary (2-4 sentences).
Input JSON: {
  "title": "<document title>",
  "abstract_or_introduction": "<abstract or introduction text>",
  "all_section_paths": [
```

```
    "<section path 1>",
    "<section path 2>",
    "<section path 3>"
  ]
}
```

3.3.4.0.5 Deep Dive Node The deep dive node is applied to parent documents that contain chunks classified as interesting but insufficient. Such a chunk is treated as information scent: it does not answer the requirement directly, but it indicates that the surrounding document may be a relevant information patch. The purpose of the deep dive node is therefore to decide whether the system should continue exploring the parent document of the interesting chunk or abandon it.

For each parent document summary derived from an interesting chunk, the system first performs a document-level relevance check. The same model, llama3:8b is used for this classification. The prompt used for the relevance gate is shown below:

```
You are a strict relevance gate.
Decide whether the document is relevant to the requirement.
Return JSON only with keys: continue_search (boolean), reason (short string).

Input JSON: {
  "query": "<user query>",
  "requirement_text": "<current requirement>",
  "summary": "<document summary>"
}
```

The summary is classified as relevant or not relevant to the requirement in order to decide whether the document should be investigated further. In other words, the system asks whether the interesting chunk provides enough scent to justify further within-patch exploration. If the document is not considered relevant, the patch is abandoned for that requirement. If the document is considered relevant, the system continues by selecting additional sections from the same document.

The next step is section selection. Instead of checking the entire document, the model selects at most two section paths that appear most likely to contain answering information. The configuration of two section paths is reasonable to limit the time and computation spent while still allowing the system to search beyond the initially retrieved chunk. The prompt used for section selection is shown below:

```
Select at most 2 section paths for follow-up chunk checking.
Return JSON only with keys: section_paths (array of <=2), reason.

Input JSON: {
  "query": "<user query>",
  "requirement_text": "<current requirement>",
  "title": "<document title>",
  "summary": "<document summary>",
  "candidate_section_paths": [
```

```
    "<candidate section path 1>",
    "<candidate section path 2>",
    "<candidate section path 3>"
  ]
}
```

The selected sections are then retrieved, and their chunks are classified and saved similarly to the classification method in Section 3.3.4.0.3, with the exception that the interesting category is removed. The purpose of the deep dive node is not to identify additional partially relevant chunks, but to determine whether the document contains chunks that directly answer the current requirement. Therefore, chunks are only classified as answering or unrelated at this stage. If additional chunks are merely interesting, they are not included in the final result, since they may add context but do not directly answer the requirement. Including such chunks would risk introducing unnecessary information and increasing the amount of context passed forward without improving the final answer.

3.3.4.0.6 End Node At this node the system has carried out the search for all requirements given in the task. The system has had the chance to fill the state with three pieces of information: answering chunks, interesting chunks and document summaries. Now the end nodes is tasked with building the following coverage result to be returned to the Knowledge Orchestrator:

```
class CoverageResult(BaseModel):
    requirementCoverage: dict[str, float]
    evidenceByRequirement: dict[str, list[str]]
```

For each requirement, the end node examines all chunks that have been retrieved and classified for that requirement in order to determine requirementCoverage. If at least one chunk is classified as `answers`, the requirement receives a coverage score of 1.0, representing full evidence. If no answering chunks are found, but one or more chunks are classified as `interesting`, the requirement receives a coverage score of 0.5, representing partial evidence. If neither answering nor interesting chunks are found, the coverage score is set to 0.0.

At the same time, the end node constructs evidenceByRequirement. This field contains a list of evidence strings for each requirement. The evidence strings are built from the motives and document summaries connected to both answering and interesting chunks. In other words, whenever a chunk is classified as either `answers` or `interesting`, its classification motive and the corresponding document summary are included in evidenceByRequirement. The motive explains why the LLM considered the chunk relevant to the requirement, while the document summary provides broader context from the source document.

3.4 System Configuration and Hyperparameters

To ensure the reproducibility of the study, this section details the technical configurations and hyperparameters used in the system. The parameters, including model selections and specific system settings, are summarized in the Table 3.1 below.

Subsystem	Parameter	Value
Knowledge Orchestrator (KO)	Max Iterations	6
	Min Coverage to Finish	0.85
Models & Embeddings	Primary LLM Model	`llama3:8b`
	Embedding Model	`nomic-embed-text`
	Embedding Dimensions	768
	Embedding Batch Size	64
Document Ingestion	Chunk Size (Tokens)	512
	Chunk Overlap (Tokens)	64
	Marker Extraction Mode	balanced

Table 3.1: Key configuration parameters for the Autonomous Knowledge Agent system.

The specific system prompts provided to the LLM to govern its reasoning and behavior during the different stages of the system can be found in Appendix B.

4
Results

This chapter presents a feasibility study of the autonomous knowledge agent, evaluating its performance based on a grey-box approach consisting of the the evaluation of the generated topic report and a couple important internal pipeline components. The topic report is evaluated in regards to three core metrics: scope, correctness and usefulness. The internal components is evaluated in regards to how well they achieve their task. The evaluation data is derived by evaluating the system's behavior across three distinct queries. This gives us enough data to highlight the strengths and weaknesses of the data system.

4.1 Evaluation Approach and Feasibility Study

Because the system was evaluated on a limited set of three queries, this evaluation is framed as a feasibility study rather than a comprehensive statistical benchmark. The goal is to demonstrate the system's functionality and identify points of failure in the data pipeline and agentic reasoning.

Evaluation of the system is based on three metrics:

- **Scope:** Measures whether the report covers all key themes generated during the requirement phase and whether it stays within the boundaries of the provided sources.
- **Correctness:** Determines whether the report accurately reflects the data in the sources without introducing hallucinations or mismatched claims.
- **Usefulness:** Determines whether the report provides specific technical insights and concrete details rather than generic general information.

The evaluation follows a grey-box approach, since it assesses both the final generated topic reports and selected internal pipeline outputs, including retrieved papers, ingestion results and requirement coverage. This allows the evaluation to connect observed report-level weaknesses to specific stages of the system, such as document ingestion, chunk classification, or final synthesis. The topic report scoring is then validated through a dual-verification process, where both a state-of-the-art LLM, Gemini 1.5 Pro, and a PhD student review the generated topic reports and score them according to the metrics above. Gemini 1.5 Pro was selected for the LLM-assisted evaluation because it represents a model with advanced reasoning and text-evaluation capabilities. Using a capable model for this role reduces the risk that the automated evaluation is limited by a weak evaluator rather than by the quality of the generated reports themselves.

The topic report scoring is then validated through a dual-verification process, where both a state-of-the-art LLM (Gemini 1.5 PRO) and an PhD student reviews the generated topic reports and scores them on the metrics above.

4.2 System Evaluation

4.2.1 Internal Processes Evaluation

To evaluate the internal processes of the system, three distinct research queries representing different domains were submitted to the Knowledge Orchestrator. For each query, the system's document search, ingestion, and chunk retrieval processes were evaluated.

Query 1: Artificial Intelligence and Machine Learning in Agriculture

The system generated 7 specific requirements:

1. Explain the applications of deep learning algorithms in precision agriculture, focusing on yield prediction and crop monitoring.
2. Describe the role of natural language processing in improving farm-to-table supply chain management through enhanced labeling and inventory tracking.
3. Analyze the potential of reinforcement learning in optimizing irrigation systems for water conservation and efficient resource allocation.
4. Discuss the current state of computer vision in detecting and classifying pests and diseases in agricultural settings, highlighting challenges and future directions.
5. Investigate the feasibility of integrating artificial intelligence with existing sensor networks to enhance real-time monitoring and predictive maintenance of farm equipment.
6. Identify concrete mechanisms, model architectures, and implementation choices used in the retrieved studies.
7. Extract quantitative performance evidence and validation setup details, including metrics, benchmarks, and measurement protocols.

- **Retrieved papers & Ingestion:** 40 candidate papers were retrieved. Nine of them were judged relevant for PDF download based on the relevance of the metadata. Two downloads from the databases failed due to unknown errors, leaving only seven papers successfully downloaded.
- **Requirement coverage:** Indicated missing information, specifically noting only "partial support" for requirements 1 and 6.

Query 2: Impact of Microplastics of Marine Ecosystems

The system generated 12 comprehensive requirements:

1. Explain how microplastics affect the structure and function of marine food chains, including their role as a trophic subsidy or predator attractant.

2. Describe the mechanisms by which microplastics influence marine ecosystem processes such as primary production, nutrient cycling, and decomposition.
3. Analyze the implications of microplastic ingestion on the physiology, behavior, and survival of marine organisms across different taxonomic groups.
4. Discuss the potential cascading effects of microplastic pollution on marine ecosystem services, including fisheries, coastal protection, and carbon sequestration.
5. Summarize the current state of knowledge on the fate and transport of microplastics in marine ecosystems, including their persistence, mobility, and bioaccumulation.
6. Retrieve experimental data on the sedimentation rates of microplastics in different marine environments to inform volume estimates.
7. Utilize remote sensing techniques and modeling tools to quantify the spatial extent and distribution of microplastic pollution.
8. Retrieve sedimentation rate data for microplastics in different marine environments to inform volume estimates.
9. Estimate the total amount of microplastic particles suspended in seawater by analyzing water samples with particle counters or microscopy.
10. Gather additional contextual and comparative background information.
11. Identify concrete mechanisms, model architectures, and implementation choices used in the retrieved studies.
12. Extract quantitative performance evidence and validation setup details, including metrics, benchmarks, and measurement protocols.

- **Retrieved papers & Ingestion:** 13 candidate papers were retrieved. However, the ingestion pipeline experienced a hardware failure, managing to download only one paper successfully.
- **Requirement coverage:** Indicated "partial support" for 10 requirements with explicit flaws in multi-source support and quantitative evidence depth.

Query 3: Influence of Sleep Quality on Cognitive Performance

The system generated 7 requirements:

1. Explain how different aspects of sleep quality (e.g., duration, fragmentation, latency) impact cognitive performance in tasks that require attention, processing speed, and memory consolidation.
2. Discuss the relationship between sleep stage patterns and memory formation, including the role of REM and non-REM sleep phases.
3. Analyze the effects of poor sleep quality on working memory capacity and executive function, including planning, decision-making, and problem-solving abilities.
4. Describe how sleep quality influences the consolidation of new information from short-term to long-term memory, including the potential for sleep-dependent memory reconsolidation.
5. Explain how individual differences in sleep quality and cognitive performance impact daily functioning, academic or professional productivity, and overall well-being.

6. Specify the experimental design and data analysis procedures used to quantify sleep quality metrics and cognitive performance outcomes.
7. Provide detailed descriptions of the statistical methods employed to control for potential confounding variables.

- **Retrieved papers & Ingestion:** 30 candidate papers were retrieved. The system successfully indexed five highly relevant papers without any download failures.
- **Requirement coverage:** Indicated "partial support" for Requirement 7 regarding detailed statistical methods.

4.2.2 Topic Report Evaluation

To evaluate the topic report the LLM and the PhD student separately carried out an evaluation of the three topic reports that was the result of the three queries above. The prompt for the LLM can be read in Appendix B. The human feedback is summarized below:

> *"Topic Report 1 (AI in Agriculture) was mostly accurate but quite generic; it lacks concrete AI methods and stronger technical evidence. Topic Report 2 (Microplastics) missed the main user query. It focuses on a detection method rather than broader ecosystem impacts. Topic Report 3 (Sleep Quality) is too narrow, focusing mainly on NREM theta power while missing broader executive function and confounding factors."* — PhD Student.

Notably, the LLM and the PhD student's assessment resulted in an identical scoring for the metrics: correctness, usefulness and scope. The identical scores confirm the credibility of the assessment of the metrics and are shown in Table 4.1 (on a scale of 1 to 5).

Table 4.1: LLM and Human Evaluation scores for the generated topic reports

Query Topic	Correctness	Usefulness	Scope
AI and ML in Agriculture	4	2	3
Microplastics in Marine Ecosystems	4	2	1
Sleep Quality and Memory	5	3	2

4.3 Diagnosis of Identified Weaknesses

The evaluation results demonstrate a strong consensus between the LLM and the PhD student evaluators. This alignment is significant as it suggests that the system's internal relevance and evaluation logic is grounded in human-like technical judgment. Both evaluators identified the same failure modes: while the system is highly capable of maintaining *Correctness* (avoiding hallucinations), it struggles

with *Scope* and *Usefulness* when the data ingestion pipeline or chunk classification logic prioritizes general background information over deep technical evidence.

The consistent pattern across the evaluation of the three queries in Section 4.2 was ingestion bottlenecks and chunk retrieval flaws. This lead to narrow information, directly causing the scope and usefulness scores to degrade.

4.3.1 Connecting Weaknesses to the Scope Scores (Average 2/5):

The system consistently failed to address the full breadth of its own generated requirements. This is diagnosed as a failure in the ingestion pipeline. In the Microplastics query (Scope 1/5) there was a catastrophic hardware failure causing 12 out of 13 candidate papers to not be ingested, severely restricting the knowledge base. Because only one source was available, the system completely ignored the query intent (marine food chains) and instead fixated entirely on the measurement tool used in that single paper. However even when ingestion was successful, as in the sleep quality query, the final report generation neglected many sources, almost entirely ignoring earlier evidence chunks to focus on a single study regarding a specific niche subject.

4.3.2 Connecting Weaknesses to the Usefulness Scores (Average 2.3/5):

This score was caused because the generated reports were either overly fluffy or highly repetitive. In the queries regarding AI in agriculture and microplastics, the reports listed generic, high-level benefits (e.g. "IoT sensors monitor soil moisture") instead of extracting the requested quantitative metrics and specific architectures. This is diagnosed as a weakness in the chunk classifier. The classifier correctly identifies chunks that are semantically related to the topic, but it fails to prioritize deep methodological or quantitative chunks over general introduction paragraphs.

4.3.3 Connecting Weaknesses to the Correctness Scores (Average 4.3/5):

Despite failing on scope and usefulness, the system achieved high Correctness scores because the final report generator effectively grounded its output in the retrieved sources. The system did not produce major hallucinations; the technical claims it made regarding details matched the source texts exactly.

4.4 Architectural Improvements

Based on the direct diagnosis of the system's failures in Section 4.3, the following architectural improvements are proposed to target the root causes of the degraded scores:

1. **Fixing Ingestion Reliability to Improve Scope:** To address the catastrophic PDF download failures that reduced the searchable corpus (as seen in the Microplastics query), the ingestion pipeline must be made more robust. This would include automated retry logic for failed downloads, controlled shut down when hardware for subsystems crashes instead of continuing with bad data.
2. **Implementing a Multi-source Synthesis Check:** To prevent the single source behavior observed in the synthesis phase, the Orchestrator's LangGraph logic must be updated. Before final generation, a judge node should verify that evidence chunks from multiple distinct sources are integrated into the text, forcing the final topic report generator to synthesize broadly rather than fixating on a single study.
3. **Improving the Chunk Classifier and Prompting to Increase Usefulness:** To eliminate the fluff in the generated reports, the chunk classifier's prompt and category definitions must be adjusted to rate chunks of a higher scientific grade, such as chunks containing concrete mechanisms, mathematical models, and quantitative metrics, as more relevant. Currently the prompts prioritize chunks containing general background information.
4. **Expanding Source Access:** To better scope and address the English-language and open-access bias, the Agentic Document Searcher should be integrated with additional academic databases, both open and closed access, as well as support multilingual retrieval to broaden the range of information available for final report generation.
5. **Improving the Agentic Document Searcher article evaluation:** To further improve Scope, the Agentic Document Searcher's evaluation function could be enhanced by analyzing the full text of candidate articles rather than relying solely on titles and abstracts. This would mitigate the risk of "false-negatives" where relevant details are buried within the body of a paper, and "false-positives", where an abstract suggests relevance that the full paper does not support.

4.5 Societal and Ethical Aspects

A key ethical concern with the system is the risk of misinformation. LLMs are known to occasionally hallucinate or generate reasonable-sounding but incorrect information, particularly when sufficient supporting data is not available. This may lead to inaccuracies in the generated topic report, and since the system's core purpose is to write source-based topic reports, any misinformation is especially risky as it may carry an unearned sense of authority, potentially leading researchers to skip the critical step of reading the original papers. Therefore, the user must remain aware that the information in the topic report must be fact-checked and verified.

Another important aspect is the system's usability for researchers; this system may reduce the time required for finding literature and relevant information. By reducing the time needed for these usually time-consuming tasks, researchers can spend more time focusing on their own research and more advanced tasks. Although, this

comes with its own consequences, since researchers might start skipping the actual reading of papers, which as said before, risks misinformation being used in actual research since the system runs the risk of incorporation false information.

Furthermore, the system can be useful for finding niche or less prominent articles that might otherwise be overlooked. Potentially uncovering relevant work that manual searches could miss. However, the system's current design introduces a significant representational bias, since only English-language sources can be retrieved and processed. This means important research published in other languages may be excluded, leading to possible knowledge gaps in the generated reports.

However, the local-first design is beneficial for privacy, but creates an ethical issue of asymmetric benefit. Because the system requires substantial computational resources to run effectively, it is primarily accessible to well-resourced researchers with high-end hardware. This creates a disparity where those with better resources gain the advantage, while others without such hardware are excluded from the systems benefits.

Finally, the system is designed to respect the legal and ethical boundaries regarding access to academic content by only processing sources for which the user has authorized access. However, an ethical challenge arises regarding the licensing of the generated output; since the reports might synthesize licensed research articles, the resulting topic report may be subject to the same restrictive terms as the source material. This requires a high level of user awareness to ensure that the automation of research does not lead to the unauthorized distribution of proprietary or copyrighted material.

4.6 Future Work

Future development should focus on system resilience and source diversity. To address the ingestion bottleneck, a distributed processing queue (e.g., Celery or Redis) should be implemented to handle PDF parsing, preventing a single large file from crashing the entire pipeline.

Furthermore, the Agentic Document Searcher could be improved by utilizing a wider range of academic databases to address current limitations in Scope. Integrating additional repositories, both open-access and closed-source, would mitigate the potential for bias and ensure the system has access to a more diverse information base for report generation. Additionally, expanding support for non-English languages would allow the system to capture regional research and global perspectives. This broader multi-source approach would significantly enhance the scope of the final topic report.

Likewise, the Agentic RAG could be improved by improving the prompts and search capabilities to extract more detailed and relevant data from the local vector database. Enhancing the system's ability to navigate deep into document information patches, would further improve both the scope and correctness of the final topic

reports.

5
Conclusion

This thesis has presented a local-first autonomous knowledge agent for privacy-preserving literature search. The system demonstrates that it is technically possible to move from a user-defined research query to a source-grounded topic report through an agentic workflow that combines requirement generation, document search and ingestion, retrieval and relevance classification, and final synthesis. A central contribution of the work is that the research query is treated as protected information, meaning that sensitive early-stage research ideas can be processed without leaking the the idea in case it is classified as confidential to you or your organization.

The feasibility study shows that the prototype is most successful at maintaining correctness. Across the evaluated queries, the generated reports were generally grounded in the retrieved sources and did not show major hallucination problems. This indicates that the local RAG-based architecture can support reliable source-based generation when relevant documents are successfully retrieved and ingested. However, the evaluation also shows clear limitations in scope and usefulness. The system often failed to cover the full breadth of its own generated requirements and sometimes produced reports that were too general or too narrowly focused on individual sources.

The main causes of these limitations were identified as ingestion reliability, limited source coverage, and insufficient prioritization of technical and quantitative evidence during chunk classification. In particular, the microplastics query showed that the system becomes highly vulnerable when only a small part of the intended literature corpus is successfully ingested. Even when ingestion worked better, the final report generation sometimes relied too heavily on a limited subset of the available evidence.

Overall, the thesis demonstrates a working prototype and a promising architecture for autonomous, local-first literature search, but not a finished research tool. Future work should therefore focus on making the ingestion pipeline more robust, improving multi-source synthesis, refining the chunk classifier, and expanding access to a broader range of academic sources. With these improvements, the system could become a more reliable tool for researchers who need AI-supported literature search while maintaining privacy.

Bibliography

[1] Z.-Z. Li et al., "From System 1 to System 2: A Survey of Reasoning Large Language Models," arXiv preprint, arXiv:2502.17419, 2025. [Online]. Available: https://arxiv.org/pdf/2502.17419

[2] W. Zhang et al., "Deep Research: A Survey of Autonomous Research Agents," arXiv preprint, arXiv:2508.12752, 2025. [Online]. Available: https://arxiv.org/pdf/2508.12752

[3] J. Liu et al., "A Comprehensive Survey on Long Context Language Modeling," arXiv preprint, arXiv:2503.17407, 2025. [Online]. Available: https://arxiv.org/pdf/2503.17407

[4] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," arXiv preprint, arXiv:2005.11401, 2020. [Online]. Available: https://arxiv.org/pdf/2005.11401

[5] R. Van Noorden and J. M. Perkel, *AI and science: What 1,600 researchers think*, Nature, vol. 621, no. 7980, pp. 672–675, 2023. [Online]. Available: https://www.nature.com/articles/d41586-023-02980-0

[6] Y. Shanmugarasa, M. Ding, M. A. P. Chamikara, and T. Rakotoarivelo, "SoK: The Privacy Paradox of Large Language Models: Advancements, Privacy Risks, and Mitigation," in Proc. ACM Asia Conf. on Computer and Communications Security (ASIA CCS '25), Hanoi, Vietnam, Aug. 2025. [Online]. Available: https://arxiv.org/html/2506.12699v1

[7] Avidnote, "Avidnote - AI for Research Writing, Reading & Analysis," [Online]. Available: https://avidnote.com/

[8] European Commission, Directorate-General for Research and Innovation, "Living guidelines on the responsible use of generative AI in research," 2nd ed., Apr. 2025. [Online]. Available: https://research-and-innovation.ec.europa.eu/document/download/2b6cf7e5-36ac-41cb-aab5-0d32050143dc_en?filename=ec_rtd_ai-guidelines.pdf

[9] ResearchRabbit, "Step-by-Step Guide to Using ResearchRabbit," [Online]. Available: https://www.researchrabbit.ai/articles/guide-to-using-researchrabbit

[10] SciSpace, "About Us," [Online]. Available: https://scispace.com/t/about/

[11] SciSpace, "How SciSpace Protects Your Uploaded PDFs," [Online]. Available: https://scispace.com/help/en/articles/10706845-how-scispace-protects-your-uploaded-pdfs

[12] H. Piwowar, J. Priem, V. Larivière, J. P. Alperin, L. Matthias, B. Norlander, A. Farley, J. West, and S. Haustein, "The state of OA: a large-scale analysis

of the prevalence and impact of Open Access articles," *PeerJ*, vol. 6, p. e4375, 2018. doi: https://doi.org/10.7717/peerj.4375

[13] P. Pirolli and S. Card, "Information Foraging," *Psychological Review*, vol. 106, no. 4, pp. 643–675, 1999. [Online]. Available: https://www.researchgate.net/profile/Peter-Pirolli/publication/229101074_Information_Foraging/links/02bfe50f098acc0ea8000000/Information-Foraging.pdf

[14] ResearchRabbit, "Data Responsibility: TLDR," ResearchRabbit Help Center, Nov. 2025. [Online]. Available: https://learn.researchrabbit.ai/en/articles/12796770-data-responsibility-tldr

[15] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W. Yih, "Dense Passage Retrieval for Open-Domain Question Answering," [Online]. Available: https://arxiv.org/abs/2004.04906

[16] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large Language Models: A Survey," arXiv preprint, arXiv:2402.06196, 2024. [Online]. Available: https://arxiv.org/pdf/2402.06196

A

Appendix

A.1 Terminology List

- **LLM**: Large Language Model is a type of artificial intelligence trained on large amounts of text data to understand, generate, and manipulate human language by predicting the next token in a sequence.
- **RAG**: Retrieval-Augmented Generation is an architectural framework that optimizes LLM output by querying an external knowledge base (often a vector database) for relevant information before generating a response, ensuring accuracy and reducing hallucinations.
- **Agentic RAG**: A RAG system that performs multi-step reasoning and tool use
- **LangGraph's StateGraph**: A framework for building stateful, cyclic workflows that allow an agent to manage iterative reasoning, branching decisions, and current state across multiple steps.
- **Semantic search**: A retrieval method that uses vector embeddings to find information based on conceptual meaning and context, rather than exact word matches.
- **Full-text search**: A retrieval method that matches explicit keywords and terms from a query against a document.
- **Local-first logic**: A design choice where all computations are hosted locally to ensure data privacy.
- **Chatbot-based**: A system where an LLM typically acts as a general purpose assistant in a chat-looking type of user interface.
- **State-of-the-art LLM**: A frontier-level Large Language Model representing the most advanced capabilities currently available in natural language understanding and generation.

B

Appendix

B.1 System Prompts

- **Keywords extraction for arXiv in Agentic Document Searcher**:

```
title_query_prompt = (
    """
    [STRICT MODE]
    TASK: Extract the 3-5 most important
    technical keywords from the USER_QUERY.

    OUTPUT FORMAT:

    Exactly 3-5 words.
    Separated by a single space.
    No punctuation.
    No introductory text.
    No numbers.

    USER_QUERY:
    """
)
```

- **Keyword extraction for OpenAlex in Agentic Document Searcher**:

```
openalex_keyword_prompt = (
    """
    You extract ONLY search keywords from a USER
       QUERY.

    Return ONLY valid JSON.
    Do NOT write explanations.
    Do NOT write text before or after the JSON.

    JSON schema:

    {
      "keywords": [],
      "authors": []
    }
```

```
        Rules:
        - Extract 2-5 keywords.
        - Keywords must be specific scientific concepts
          (methods, diseases, technologies, compounds).
        - Prefer domain-specific terms over general
           phrases.
        - Keep multi-word terms when they represent a
           clear concept
          (e.g., "GLP-1 agonists").
        - Remove filler words such as "find", "show",
           "papers",
          "articles".
        - Do NOT include vague phrases like "current
           evidence",
          "recent studies", "analysis", "overview".
        - Do NOT include time expressions such as "last
           5 years",
          "recent", "since 2020".
        - If no author is mentioned, return [].

        USER QUERY:
        {user_query}
        """
    )
```

- **Evaluation of candidate articles for arXiv in Agentic Document Searcher**:

```
    system_prompt_evaluation = (
        "You are an expert academic reviewer. "
        "Score the relevance of a paper abstract to a
           user's research
        request on a scale of 0 to 10:\n"
        "- 10: Perfect match. Directly addresses the
           core topic and
        methodology requested.\n"
        "- 7-9: Highly relevant. Provides significant
           evidence or
        data related to the topic.\n"
        "- 4-6: Partially relevant. Mentions the topic
           or provide
        s useful background context.\n"
        "- 1-3: Low relevance. Only tangentially related
           or mentions
        keywords in passing.\n"
        "- 0: Completely irrelevant. Different field of
           study or
        unrelated subject.\n"
        "Consider the title, abstract, and findings. "
        "DO NOT provide reasoning. ONLY output the
```

```
            integer score."
    )
```

- **Evaluation of candidate articles for OpenAlex in Agentic Document Searcher**:

```
openalex_relevance_prompt = (
    f"""
    Research topic:
    {topic}

    Papers:
    {papers_block}

    For each paper, score how relevant it is to the
       research topic.

    Match the main idea, not just shared words.
    A paper can still be relevant if it uses different
       words but has
    the same core idea, problem, method, or application.
    Synonyms and close paraphrases count as matches.

    Do not score high for broad or generic matches only.

    Return ONLY a list of exactly {len(batch)} integers
       between 1 and 5.
    Example: [4,2,1,5,3]

    Scoring:
    1 = unrelated
    2 = weak relation
    3 = partial to moderate relevance
    4 = strong relevance
    5 = highly relevant
    """
)
```

- **Requirements extraction in Knowledge Orchestrator**:

```
    prompt = (
        f"""Analyze the user query and extract up to 5
           specific thematic content requirements that
           will serve as the sole criteria for judging a
           scientific report's completion.

The Evaluation Context:
The system evaluating the final report will not have
   access to the original query. It will only see these
   requirements. Therefore, each requirement must be a
   descriptive, self-contained instruction regarding the
   scientific subject matter.
```

```
Strict Constraints:

Content Only: Focus exclusively on technical themes,
   phenomena, or specific scientific arguments.

No Metadata: Ignore source types, dates, authors, or
   publication constraints.

Descriptive Clarity: Avoid one-word requirements. Use
   phrases that clearly define what the report must
   explain or analyze.

Return one requirement per line without numbering.

Query: {query}"""
    )
```

- **Diagnosis and re-planning of requirements in Knowledge Orchestrator**:

```
    prompt = (
        "You are a strict scientific research planner.\n"
        "Your job is to diagnose what evidence
           dimensions are still missing and propose
           improved retrieval requirements.\n"
        "Output valid JSON only, with this schema:\n"
        "{\n"
        '  "missing_dimensions":
           ["methods"|"quantitative"|"comparison"|"measurement"|
        "implementation"|"source_diversity"],\n'
        '  "requirements": ["...", "..."]\n'
        "}\n"
        "Constraints:\n"
        "- Propose 1 to 3 requirements only.\n"
        "- Each requirement must be specific and
           technical, max 140 chars, no numbering.\n"
        "- Do not repeat existing requirements
           semantically.\n"
        "- Focus only on missing dimensions indicated by
           the profile and evidence.\n\n"
        f"Current requirements: {json.dumps([r.text for
           r in ko_state.requirements],
           ensure_ascii=True)}\n"
        f"Depth deficiency profile: {json.dumps(profile,
           ensure_ascii=True)}\n"
        f"Evidence preview:
           {json.dumps(evidence_preview,
           ensure_ascii=True)}\n"
    )
```

- **Final report generation in Knowledge Orchestrator**:

```
prompt = (
    "You are writing the final user-facing topic
       report for a scientific research assistant.\n"
    "Write a clear, grounded markdown report that
       actually explains the topic using only the
       provided evidence.\n"
    "Do not mention orchestration, requirements,
       coverage percentages, debug notes, lexical
       scores, chunk ids, or internal pipeline
       behavior in the body.\n"
    "Do not say that this is a draft.\n"
    "Paraphrase the evidence into readable prose. Do
       not copy long equations, OCR artifacts,
       figure captions, or image markup.\n"
    "Treat the provided outline as a minimum content
       contract. Preserve its concrete distinctions
       rather than collapsing them into generic
       themes.\n"
    "If the evidence names specific actors, schools,
       methods, typologies, governance frameworks,
       or corpus details, mention those exact names
       in the prose.\n"
    "Prefer analytical synthesis over generic
       summary. Compare positions, highlight
       disagreements, and preserve methodological
       specificity.\n"
    "If a section has evidence, synthesize it with
       technical depth. Name methods, architectures,
       and experimental setup details where
       provided.\n"
    "If a section lacks evidence, either omit that
       section or mention briefly that the available
       literature in this run did not provide enough
       direct evidence.\n"
    "Be honest about limits, but do not dominate the
       report with caveats.\n"
    "Use markdown with this structure exactly:\n"
    "# Topic Report\n"
    "## Overview\n"
    "3-5 paragraphs explaining the main topic,
       technical methods, measured outcomes, and why
       the topic matters.\n"
    "## Scope\n"
    "One short paragraph on the review scope and
       recency framing.\n"
    "Then one '##' section per requirement heading
       in the provided order, using each provided
       heading exactly.\n"
```

```
            "For each requirement section, include:\n"
            "1) Methods/Mechanisms, 2) Quantitative
               Evidence, 3) Limitations and Trade-offs.\n"
            "Each requirement section must contain at least
               one sentence that starts with 'Mechanism:'
               and at least one sentence that starts with
               'Evidence:' grounded to source citations.\n"
            "Finish with:\n"
            "## Sources Consulted\n"
            "Use flat bullet points that map each provided
               source label to a Harvard-style source
               reference.\n"
            "Do not add any 'Coverage Notes' or
               system-status section.\n\n"
            "Each evidence-grounded claim should cite a
               source inline using brackets like [Source
               1].\n"
            "Only use source labels that appear in the
               provided input JSON.\n"
            "Map those labels to bullets in 'Sources
               Consulted'.\n"
            "When multiple evidence items support a section,
               combine them into a coherent synthesis.\n"
            "Write in the style of a strong PhD-level
               literature overview: natural, specific,
               analytical, and not robotic.\n"
            "Do not produce a high-level executive summary
               style output.\n"
            "Reject generic filler such as 'societal values,
               technological advances, and regulation'
               unless the evidence itself makes that exact
               argument.\n\n"
            f"Input JSON:\n{json.dumps(payload,
               ensure_ascii=True, indent=2)}"
        )
```

- **System prompt for topic report evaluation using GEMINI PRO**:

```
Role: You are an expert academic peer reviewer and
   technical auditor specializing in Cyber-Physical
   Systems and AI.

Task: Evaluate the provided "Topic Report" based on
   the three metrics defined below. You must
   cross-reference the reports claims against the
   "Sources Consulted" and "Evidence Trace" provided
   at the end of the report.

Metrics:
```

```
Correctness (Accuracy & Grounding): Does the report
   accurately reflect the data in the sources? Are
   there "hallucinations" (claims made that aren't
   in the sources) or "mismatches" (using a sensor
   source to justify an antenna claim)?

Usefulness (Depth & Insight): Does the report
   provide specific, actionable technical insights,
   or is it filled with "fluff" and generic
   generalizations? Would a professional in this
   field find this summary valuable?

Scope (Coverage & Relevance): Did the report cover
   all key themes mentioned in the Evidence Trace?
   Did it stay within the boundaries of the provided
   sources, or did it wander into unrelated topics?

Scoring Scale: 1 to 5 for each metric (1 = Poor, 5 =
   Exceptional).



Output Format:



Executive Summary: A 2-sentence overview of the
   report's quality.

Metric Table: A table showing the scores for
   Correctness, Usefulness, and Scope.

Evidence Audit: List any specific claims in the
   report that are NOT supported by the provided
   source list.

Target Level: Identify if the report reads like an
   Elementary, Undergraduate, or PhD-level document.

Key Improvement: The single most important change
   needed to make the report more professional.
```

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY