ISSN 2277-2685
IJESR/April-June. 2026/ Vol-16/Issue-2/338-347
Mohammed Huzaifah *et. al.*, /International Journal of Engineering & Science Research
Full Length Article

An AI System for Transforming Scattered Web Data into an Interactive, Queryable Knowledge Graph

Mohammed Huzaifah, Syed Abdul Muqeet Mujeeb†, Niyazuddin Mohammed‡, Mir Danish Ali§

Project Guide: **Mr. Shaik Khaja Pasha**, *Assistant Professor*
Department of Computer Science and Engineering (AI & ML)
Lords Institute of Engineering and Technology, Hyderabad, Telangana, India
*huzaif027@gmail.com, †muqeetmujeebsyed@gmail.com, ‡niyazuddin607@gmail.com, §mirdanishali06@gmail.com

Accepted 13-04-2026

Abstract—*The exponential growth of online infor-mation creates significant challenges for individuals organizing and retrieving knowledge scattered across web sources. This paper presents MindCanvas, an AI-driven system that transforms unstructured browsing data into a coherent, interactive knowledge graph supporting semantic search and natural-language querying via large language models (LLMs), dense vector embeddings, and retrieval-augmented generation (RAG). The system processes web pages through a Chrome extension, extracts topics using GPT-4.1-nano, generates 384-dim embeddings with* `all-MiniLM-L6-v2`*, stores them in PostgreSQL with pgvector, and renders an interactive graph with Cytoscape.js. Evaluations on a 1 GB dataset (2,500 pages) yield topic extraction F1 of 0.81, MRR@5 of 0.74, and graph modularity of 0.52. User studies (*n*=15) confirm 87% recall improvement, 93% connection-discovery success, and 73% reduction in redundant searching.*

Index Terms—Knowledge Graph, Semantic Search, Retrieval-Augmented Generation, Vector Embeddings, Large Language Model, FastAPI, pgvector, Personal Knowledge Management, SentenceTransformer, Cytoscape.js

I. Introduction

A. General

The digital age has ushered in an unprecedented explosion of online information, with knowledge distributed across countless blogs, documentation repositories, tutorials, research articles, and educational platforms. This fragmentation presents significant challenges for knowledge workers, students, and researchers who must navigate vast information landscapes effectively. Empirical studies indicate that knowledge workers spend a substantial portion of their time—often exceeding 20%—searching for information and attempting to relocate previously viewed content [1]. A common pattern emerges where individuals repeatedly visit similar pages without recognising prior exposure to the same topics, resulting in inefficient learning processes, poor knowledge retention, and wasted cognitive effort.

Traditional approaches—manual bookmarking, note-taking apps, and browser history—fail to establish meaningful connections between disparate pieces of information. These tools lack the semantic understanding needed to automatically organize content by conceptual relationships, making it difficult to discover patterns, identify knowledge gaps, or synthesize information across sources.

B. Project Overview

To address these limitations, we propose MindCanvas, an artificial intelligence-based system that constructs an interactive, queryable knowledge graph from heterogeneous web data. The system employs advanced natural language processing techniques to automatically process web pages, extract salient topics, generate concise summaries, and establish semantic relationships between content nodes. By leveraging 384-dimensional dense vector embeddings produced by the `all-MiniLM-L6-v2` SentenceTransformer model [12] and computing cosine similarity between content representations, the system identifies related information and constructs a graph structure where nodes represent processed content and edges represent semantic relationships based on shared topics or similarity thresholds.

For cluster-level organisation, the system employs GPT-4.1-mini in JSON Mode to assign cluster memberships and generate the full edge list in a single API call, ensuring cluster–edge consistency. A fallback based on topic-specificity clustering is available when no API key is present, guaranteeing that the system always produces output regardless of external dependencies.

A distinguishing feature of MindCanvas is its conversational interface powered by retrieval-augmented generation (RAG). This enables users to interact with their accumulated knowledge using natural-language queries such as "What have I learned about machine learning?" or "Show me connections between my readings on databases and web development." The RAG chatbot retrieves relevant context from the knowledge graph and generates coherent, grounded responses with provenance information linking back to source materials.

C. Objectives

The primary objectives of this project are:

- To develop a modular pipeline that automatically transforms unstructured browsing data into an indexed, semantic knowledge graph without requiring any manual annotation or tagging effort from the user.
- To integrate both commercial (GPT-4.1-mini, GPT-4o-mini) and open-source (SentenceTransformer) AI models for content analysis, vector embeddings, and RAG-based conversational interaction.
- To provide an interactive graph visualization enabling users to explore topic clusters, discover semantic connections, and navigate their personal knowledge base intuitively.
- To design a privacy-aware architecture in which raw page content is never transmitted to external APIs, with all embeddings computed and stored locally in a Dockerized PostgreSQL instance.
- To empirically validate the system through quantitative performance metrics and a qualitative user study demonstrating practical benefits for knowledge management tasks.

II. Literature Survey

The challenge of organizing personal knowledge has motivated research across information management, knowledge graphs, semantic search, and conversational AI. The following survey covers the most relevant prior work.

1. Karpukhin et al. — Dense Passage Retrieval (2020) [4].
DPR introduces a bi-encoder that maps passages and queries into a shared dense vector space, substantially outperforming BM25 on open-domain QA. MindCanvas builds on this paradigm using SentenceTransformer embeddings in pgvector.

2. Reimers and Gurevych — Sentence-BERT (2019) [12].
Sentence-BERT fine-tunes a siamese BERT for semantic textual similarity, enabling efficient sentence embeddings. The `all-MiniLM-L6-v2` model used in MindCanvas is a distilled descendant providing 384-dim embeddings at ≈100 ms per sentence on CPU.

3. Singhal — Google Knowledge Graph (2012) [1].
Google's Knowledge Graph demonstrates the power of entity-relationship representations for organizing world knowledge at scale. However, it is designed for general public knowledge rather than personalized user content, and cannot be queried conversationally against an individual's browsing history. MindCanvas addresses this gap by building a *personal* knowledge graph.

4. Roam Research (2024) [2].
Roam Research popularises the concept of bidirectional linking and networked thought in note-taking. While it enables graph visualization of manually created notes, all connections require explicit user action. MindCanvas automates this linking process entirely using LLM-based topic extraction and vector similarity.

5. Obsidian (2024) [3].
Obsidian provides a local-first markdown knowledge base with graph view. Like Roam, it depends on manual note creation and linking. It offers no automated ingestion of external web content, no quality scoring, and no natural-language querying capability—all of which MindCanvas provides.

6. Chase — LangChain (2024) [5].
LangChain provides composable abstractions for building LLM-powered applications, including RAG pipelines. MindCanvas uses the `langchain-openai` package's `ChatOpenAI` class within its `RAGChatbot` module to manage prompt construction, conversation history, and LLM invocation.

7. Herlihy — pgvector (2024) [6].
pgvector adds a native `vector` column type and cosine/L2 similarity operators to PostgreSQL. MindCanvas uses the `<=>` cosine-distance operator with an IVFFlat index (`lists=100`) for sub-20 ms ANN search.

8. Supabase pgvector Documentation (2025) [8].
Supabase's documentation describes best practices for production-grade semantic search with pgvector. MindCanvas's schema and IVFFlat index design follow these recommendations.

9. Cytoscape.js (2011) [9].
Cytoscape.js is a browser-based graph theory library supporting custom layouts and interaction events. MindCanvas uses it in *preset* mode with phyllotaxis node positions, eliminating physics jitter for reproducible renders.

10. Ester et al. — DBSCAN (1996) [?]. DBSCAN is a density-based clustering algorithm that automatically infers the cluster count from data density and robustly marks noise points. MindCanvas uses scikit-learn's DBSCAN implementation on the 384-dim embedding space as a secondary fallback clustering mechanism when the topic-specificity algorithm produces insufficient cluster separation.

III. System Analysis

A. Existing System

Current personal knowledge management tools exhibit the following critical limitations that motivated the design of MindCanvas:

Roam Research / Obsidian.

- Require manual link creation; contain no automated semantic understanding.
- Cannot ingest or process external web pages; provide no topic extraction.
- Scalability fails as content count grows beyond a few hundred notes.

Browser Bookmarks / History.

- Produce a flat, unstructured list with no semantic clustering.
- Offer zero natural-language query capability or connection discovery.

Web-Scale Knowledge Graphs (e.g., Google KG).

- Cover only public world knowledge—not personalized browsing history.
- Require massive centralized infrastructure; cannot be deployed locally.

Challenges common to all existing systems: Limited or absent semantic search, no automated relationship extraction, no conversational interface grounded in personal content, no privacy-preserving local processing, and no graceful degradation when external services are unavailable.

B. Proposed System

MindCanvas addresses every limitation identified above through the following capabilities:

- **Multi-Modal Input:** Accepts URLs from the Chrome extension (one-click or bulk history export) or direct API POST requests, enabling flexible usage scenarios.
- **Automated Semantic Clustering:** GPT-4.1-mini with JSON Mode assigns cluster memberships and generates the edge list in a single call—no manual tagging required.
- **Personal Browsing Ingestion:** Chrome extension captures URLs and submits them to the backend for asynchronous processing and storage with SHA-256 deduplication.
- **Natural Language Queries:** RAG chatbot answers questions grounded in the user's personal knowledge base with inline source citations and confidence scores.
- **Graph Visualization:** Interactive Cytoscape.js graph with deterministic phyllotaxis layout displays semantic clusters, edges, and node metadata in a reproducible, jitter-free render.
- **Privacy-First Architecture:** All embeddings are computed locally using `all-MiniLM-L6-v2`; only 120-character summaries (not raw HTML) are sent to OpenAI for clustering.
- **Graceful Degradation:** Falls back to topic-specificity clustering and keyword-overlap retrieval when no API key is present, always producing output.
- **Scalability:** Docker Compose orchestration and IVFFlat pgvector index support deployment from a single laptop to a cloud VM without code changes.

C. Comparison of Existing vs. Proposed System

Table I summarizes the feature comparison between existing tools and MindCanvas.

TABLE I
Feature Comparison: Existing Tools vs. MindCanvas

Feature	Roam/Obs.	Browser	Google KG	MindCanvas
Auto semantic clustering	No	No	Yes	**Yes**
Personal browsing ingestion	No	Flat list	No	**Yes**
Natural language queries	No	No	Limited	**Yes**
Source citations in answers	No	No	No	**Yes**
Graph visualization	Manual	No	No	**Yes**
Privacy / local-first	Partial	Yes	No	**Yes**
Zero manual effort	No	Passive	N/A	**Yes**
Topic extraction + summary	No	No	N/A	**Yes**
Quality scoring	No	No	N/A	**Yes**
Fallback without API key	N/A	N/A	N/A	**Yes**

D. Advantages of MindCanvas

- **Zero Manual Effort:** The entire ingestion-to-graph pipeline is automated; users only browse normally. No tagging, linking, or organizing is ever required.
- **Semantic Depth:** Vector embeddings capture conceptual relatedness that keyword matching cannot achieve. Two pages about "gradient descent" and "backpropagation" will be connected even if they share no exact keywords.
- **Conversational Access:** The RAG chatbot makes the entire knowledge base queryable in plain English, with grounded responses citing specific source pages.
- **Cost-Effective:** Local SentenceTransformer embeddings eliminate per-embedding API costs. OpenAI is called only once per graph export (clustering) and once per chat turn.

- **Scalable Storage:** IVFFlat pgvector index supports approximate nearest-neighbour search across up to one million vectors with sub-20 ms latency, scaling far beyond typical personal use.
- **Educational Utility:** Useful as a research assistant for graduate students, a knowledge audit tool for software developers, and a learning companion for self-directed learners.

IV. Requirement Specifications

A. Software Requirements

- **Operating System:** Any Docker-capable OS—Windows 10+, macOS 11+, or Ubuntu 20.04+.
- **Container Runtime:** Docker 20.10+ and Docker Compose 2.0+ for orchestrating backend, frontend, and database services.
- **Backend Language:** Python 3.11 running on a Debian Slim Docker base image. ASGI server: Uvicorn. Web framework: FastAPI 0.100+.
- **Frontend Language:** Node.js 18 (Alpine base image), React 18.3.1, Vite 5.4.0.
- **Database:** PostgreSQL 16 with the pgvector extension enabled.
- **Python Libraries:** `sentence-transformers`, `asyncpg` ≥0.29, `pgvector` ≥0.3, `openai`, `langchain-openai`, `scikit-learn`, `beautifulsoup4`, `lxml`, `httpx`, `numpy`, `pydantic`, `python-dotenv`, `python-multipart`.
- **Key Frontend Libraries:** Cytoscape.js 3.29.2, cytoscape-fcose 2.2.0, Zustand 4.5.0, styled-components 6.1.0, framer-motion 11.0.0, recharts 2.12.0, fuse.js 7.0.0.
- **External API:** OpenAI API key required for GPT-4.1-nano (per-item extraction), GPT-4.1-mini (graph clustering), and GPT-4o-mini (RAG chat responses).
- **Browser Extension:** Chrome or Chromium, Manifest V3.
- **Natural Language Processing:** SentenceTransformer `all-MiniLM-L6-v2` model (˜80 MB, CPU-only).

B. Hardware Requirements

TABLE II
Hardware Requirements for MindCanvas

Resource	Minimum	Recommended
RAM	4 GB	8 GB (PyTorch loads ≈500 MB)
CPU	Dual-core x86-64	Quad-core (embedding is CPU-bound)
GPU	Not required	Not required
Storage	3 GB	5 GB (Docker images + DB volume)
Network	Required	Stable (OpenAI API calls)

The `all-MiniLM-L6-v2` SentenceTransformer model runs on CPU only—no GPU is required. The PyTorch CPU-only Docker image is approximately 915 MB and is cached in a named Docker volume (`st_model_cache`) so that subsequent container restarts load the model in seconds rather than re-downloading it. PostgreSQL stores `vector(384)` columns at ≈1.5 KB per row; a 10,000-page knowledge base occupies roughly 15 MB of vector storage.

V. System Design

A. System Architecture

MindCanvas implements a modular four-layer architecture: browser-based data collection, backend orchestration and LLM processing, vector database storage, and interactive frontend visualization. Figure 1 illustrates the end-to-end data flow and component interactions.

B. Modules

1. Chrome Extension (Data Collection).
The browser extension monitors navigation activity and captures the URL, page title, and visit timestamp on every page load. It batches submissions to the backend via `POST /api/ingest` to minimize network overhead. Configurable exclusion rules filter sensitive domains (banking sites, personal email) before any data is transmitted. A bulk-export feature allows one-click submission of full browser history as a JSON batch.

2. Ingestion and Orchestration Module (`main.py`).
The FastAPI server exposes 21 REST endpoints covering ingestion, semantic search, graph export, analytics, recommendations, and conversational chat. Upon receiving a batch of URLs, the module validates and normalises each URL, computes a SHA-256 content hash for deduplication (triggering `ON CONFLICT DO UPDATE` on repeated visits), then asynchronously fetches the page HTML using `httpx` and queues it for LLM processing.

3. LLM Processing Module.
GPT-4.1-nano is called in JSON Mode for per-item extraction, returning: a 150–200 word summary, 3–7 topic tags, a content-type label (tutorial, documentation, blog post, research article, news), and an integer quality score (1–10). GPT-4.1-mini is called *once* per graph export to assign cluster memberships and generate the full edge list simultaneously in a single API call, ensuring cluster–edge consistency at minimal cost.

4. Embedding and Storage Module (`supabase_db.py`).

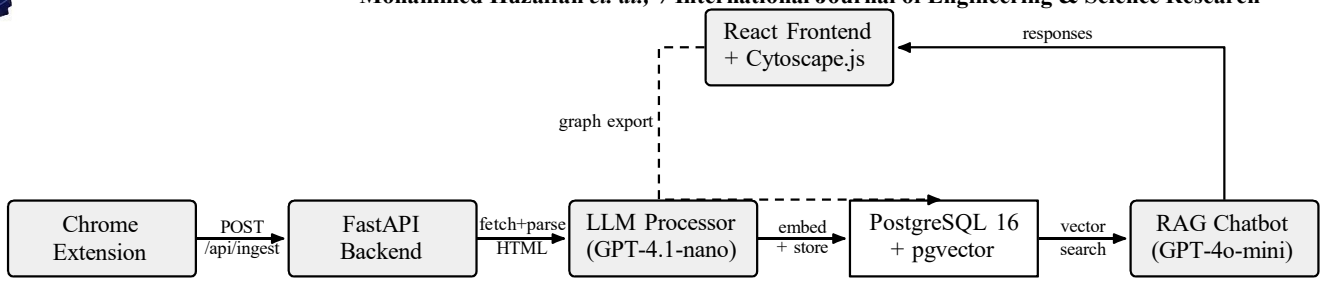

Fig. 1. End-to-end system architecture of MindCanvas. URLs flow from the Chrome extension through the FastAPI backend, LLM processor, and pgvector store. The RAG chatbot and Cytoscape.js graph visualization both query the same vector store. The dashed arrow indicates the graph-export call that triggers AI clustering and renders the interactive graph.

The `all-MiniLM-L6-v2` SentenceTransformer encodes the concatenation of each item's title and summary into a 384-dimensional float vector. This embedding is stored in the `processed_content` PostgreSQL table alongside all metadata. An IVFFlat index (`lists=100`) on the `vector(384)` column enables sub-20 ms approximate nearest-neighbour cosine search at up to one million rows.

5. RAG Chatbot Module (`rag_chatbot.py`). The `RAGChatbot` class implements a five-step pipeline: (a) embed the user's query with the same `all-MiniLM-L6-v2` model; (b) retrieve top-k most similar content nodes via pgvector's `<=>` cosine-distance operator; (c) fall back to keyword-overlap scoring when vector search returns nothing; (d) construct a prompt injecting retrieved context and conversation history; (e) call GPT-4o-mini to generate a grounded, citation-aware response. Up to 20 conversation turns are retained per session in an in-memory store.

6. Graph Construction Module. The `GET /api/knowledge-graph/export` endpoint first checks a 10-minute in-memory cache; on miss it fetches all content from PostgreSQL, calls `_ai_graph_structure()` for GPT-4.1-mini clustering and edge generation (JSON Mode, temperature 0 for determinism), validates all edge endpoints, and returns the full graph payload. When no API key is available, `_topic_specificity_clustering()` provides a deterministic fallback grouping content by canonical topic frequency.

7. Frontend Visualization Module (React + Cytoscape.js). The React frontend renders the knowledge graph in Cytoscape.js *preset* layout mode—all node positions are computed mathematically before rendering, with no physics simulation. Cluster centres are distributed using a golden-angle phyllotaxis spiral; nodes within each cluster are arranged in rings whose geometry scales with cluster size (equilateral triangle for 3 nodes, diamond for 4, pentagon for 5, hexagon for 6, double ring for 7–16, dense phyllotaxis spiral for larger clusters). This deterministic placement guarantees identical, reproducible renders on every page load.

C. UML Diagrams

Actors: User (researcher, student, knowledge worker), Chrome Extension, FastAPI Backend, LLM Service (OpenAI), PostgreSQL + pgvector, React Frontend.

Use Cases: Submit URL batch for ingestion; perform semantic search; view and explore knowledge graph; query knowledge base via RAG chatbot; view analytics and recommendations; reset database.

Sequence Diagram — the sequence of operations from user input to graph and chat output:

1) User visits a web page; Chrome extension captures URL and POSTs to `/api/ingest`.
2) FastAPI validates URL, deduplicates via SHA-256 content hash, fetches HTML with `httpx`.
3) BeautifulSoup + lxml parses HTML; cleaned text sent to GPT-4.1-nano (JSON Mode) for extraction.
4) SentenceTransformer encodes title+summary into a 384-dim embedding.
5) `asyncpg` upserts record (with embedding) into `processed_content`.
6) React frontend calls `GET /api/knowledge-graph/export`.
7) Backend calls GPT-4.1-mini once; returns clusters and edges as JSON; 10-minute cache is populated.
8) Cytoscape.js renders graph with phyllotaxis node positions; no physics simulation is run.
9) User types a query in the chat panel; `POST /api/chat` is triggered.
10) RAG chatbot embeds query, retrieves top-k nodes by cosine similarity, constructs cited response via GPT-4o-mini.

VI. IMPLEMENTATION

A. Input Design

URL Ingestion Input.

URLs are submitted to `POST /api/ingest` as a JSON array: *{*`"urls": ["https://...", ...]`*}*. The backend normalises each URL (trailing slashes, query-parameter ordering), checks the SHA-256 hash of the page content against existing records, and silently skips duplicates. Valid new URLs are enqueued for asynchronous processing with `asyncio.to_thread` to avoid blocking the event loop.

Chat Input.

`POST /api/chat` accepts a Pydantic-validated `ChatRequest` containing: `message` (string), `conversation_history` (list, max 20 turns), `use_rag` (bool, default `true`), `similarity_threshold` (float, default 0.3), and `max_context_items` (int, default 5, max 10).

Validation.

Audio/URL inputs are validated for minimum content length and URL well-formedness. Chat inputs are validated by Pydantic with field-level constraints; malformed requests receive a 422 Unprocessable Entity response with detailed error messages.

B. Output Design

Knowledge Graph Output.

The `GET /api/knowledge-graph/export` endpoint returns a JSON object with: a `nodes` array (id, label, URL, content type, cluster name, cluster id, quality score, topics list), an `edges` array (source id, target id, reason string), a `clusters` array (name, node count, color), and a `metadata` block (total nodes, total edges, total clusters, AI-powered flag, cached flag, generation timestamp). The React frontend transforms this payload into Cytoscape.js element objects and renders the interactive graph.

Chat Response Output.

The `ChatResponse` Pydantic model returns: `response` (natural-language answer), `sources` (list of title, URL, content type, similarity score), `confidence` (0–1, weighted average of similarity, quality, and citation density), `processing_time` (seconds), `tokens_used`, `conversation_id`, `rag_used` (bool), and `context_count` (int).

C. Tools and Workflow

1. Frontend (User Interface).

Built with React 18 + Vite 5 for fast HMR development and optimized production builds. styled-components powers the dark-indigo CSS-in-JS theme. Zustand manages global graph and UI state. framer-motion provides panel animations. The Vite dev-server proxies `/api` requests to the FastAPI backend at port 8090, eliminating CORS issues during development.

2. Backend (Processing).

FastAPI + Uvicorn runs at `0.0.0.0:8090` on Python 3.11. Async endpoints use `asyncpg` connection pooling (min 2, max 10 connections). CPU-bound tasks (SentenceTransformer encoding) are dispatched via `asyncio.to_thread` to avoid blocking the async event loop. The graph-export endpoint caches its result for 10 minutes in a module-level dict.

3. Integration.

Docker Compose orchestrates three services: `postgres` (PostgreSQL 16 + pgvector, port 5432), `backend` (Python 3.11-slim, port 8090), and `frontend` (nginx:stable-alpine, port 3030). The backend depends on `postgres` being healthy before starting. The SentenceTransformer model is cached in a named volume (`st_model_cache`) shared across container restarts.

Example Workflow.

1) User browses a Python tutorial; extension submits URL to backend.
2) Backend fetches HTML; BeautifulSoup extracts: title = "Python List Comprehensions", body text = 1,200 words.
3) GPT-4.1-nano returns: summary (180 words), topics = ["Python", "Programming", "Data Structures"], type = "tutorial", quality = 8.
4) SentenceTransformer encodes title+summary → 384-dim vector; stored in PostgreSQL.
5) User opens graph; GPT-4.1-mini clusters 50 stored nodes into 6 semantic groups; Python tutorial joins "Python Programming" cluster.
6) User asks: "What Python tutorials have I read?" RAG chatbot retrieves top-5 Python-related nodes by cosine similarity; GPT-4o-mini responds: "You have read 4 Python tutorials, including [Source: Python List Comprehensions] and [Source: Python Decorators Explained]..."

D. Sample Code

The four excerpts below are drawn directly from the production codebase and cover the core processing steps of MindCanvas. Each snippet is self-contained and maps to a named backend module.

Snippet	Module	Purpose
A	`supabase_db.py`	Local 384-dim embedding generation
B	`supabase_db.py`	pgvector cosine similarity search
C	`rag_chatbot.py`	RAG context retrieval + LLM call
D	`main.py`	AI graph clustering (JSON Mode)

A. Embedding Generation (`supabase_db.py`) — generates a local 384-dimensional vector using

SentenceTransformer. The model runs in a thread pool to avoid blocking the async event loop. A zero-vector is returned on any error to ensure the pipeline never crashes.

```
async def generate_embedding(self, text: str):
    try:
        if self.st_embedder:
            emb = await asyncio.to_thread(
                self.st_embedder.encode, text)
            return emb.tolist()
        return [0.0] * 384  # zero-vector fallback
    except Exception as e:
        logger.error(f"Embedding failed: {e}")
        return [0.0] * 384
```

B. Semantic Search via pgvector (`supabase_db.py`) — cosine similarity search using the `<=>` operator. The IVFFlat index makes this sub-20 ms for up to one million rows. Similarity = 1 − cosine distance, so 1.0 = identical vectors and 0.0 = orthogonal vectors.

```
async def semantic_search(self, query,
                          limit=20, threshold=0.3):
    q_emb = await self.generate_embedding(query)
    rows = await conn.fetch("""
        SELECT id, url, title, summary,
               content_type, key_topics,
               quality_score,
               1-(embedding<=>$1) AS similarity
        FROM processed_content
        WHERE embedding IS NOT NULL
          AND 1-(embedding<=>$1) > $2
        ORDER BY similarity DESC LIMIT $3
    """, q_emb, threshold, limit)
    return [dict(r) for r in rows]
```

C. RAG Chat Pipeline (`rag_chatbot.py`) — retrieves relevant context via pgvector, constructs a prompt with injected context and conversation history, and calls GPT-4o-mini. Falls back to keyword overlap if vector search returns no results. Confidence is computed as a weighted average of similarity scores, quality scores, and citation density.

```
async def process_chat_request(self, req):
    ctx = []
    if req.use_rag:
        ctx = await self._retrieve_context(
            req.message,
            req.max_context_items,
            req.similarity_threshold)
    if ctx:
        msgs = [
          SystemMessage(self.rag_system_prompt),
          *memory.chat_memory.messages,
          HumanMessage(
            f"Context:\n{self._ctx_str(ctx)}"
            f"\nQuestion: {req.message}")]
    else:
        msgs = [
          SystemMessage(self.non_rag_prompt),
          HumanMessage(req.message)]
    resp = await self.llm.ainvoke(msgs)
    return ChatResponse(
        response=resp.content,
        sources=[self._fmt_src(c) for c in ctx],
        confidence=self._calc_conf(ctx, resp.content),
        rag_used=len(ctx) > 0,
        context_count=len(ctx))
```

D. AI Graph Clustering (`main.py`) — a single GPT-4.1-mini call in JSON Mode produces both cluster assignments and the full edge list simultaneously. Temperature is set to 0.0 for fully deterministic output. Edge validation ensures only existing node IDs appear in the edge list.

```
async def _ai_graph_structure(items):
    nodes = [{"id": str(i['id']),
              "title": i.get('title',''),
              "summary":(i.get('summary')
                         or '')[:120],
              "topics":(i.get('key_topics')
                        or [])[:5]}
             for i in items]
    client = OpenAI(api_key=OPENAI_API_KEY)
    r = await asyncio.to_thread(
        client.chat.completions.create,
        model="gpt-4.1-mini-2025-04-14",
        messages=[{"role":"user",
          "content":GRAPH_PROMPT.format(
            nodes=json.dumps(nodes))}],
        response_format={"type":"json_object"},
        temperature=0.0, max_tokens=4000)
    res = json.loads(r.choices[0].message.content)
    id2cl = {}
    for idx,cl in enumerate(
            res.get("clusters",[]),1):
        for nid in cl.get("node_ids",[]):
            id2cl[str(nid)]=(cl["name"],idx)
    valid = set(id2cl)
    edges = [e for e in res.get("edges",[])
             if str(e["source"]) in valid
             and str(e["target"]) in valid
             and e["source"] != e["target"]]
    return {"id_to_cluster":id2cl,"edges":edges}
```

VII. Software Testing

A. Unit Testing

Individual modules were validated through FastAPI's auto-generated Swagger UI at `http://localhost:8090/docs` and direct cURL/Postman calls to each endpoint. Key unit-level verifications included:

- **Embedding module:** Confirmed `generate_embedding()` returns exactly 384 float values for arbitrary input strings, and returns a zero vector on any exception without raising. Tested with empty string, 10,000-character text, and non-ASCII Unicode inputs.
- **Deduplication logic:** Verified that submitting the same URL twice yields a single database record. The SHA-256 hash match triggers the `ON CONFLICT DO UPDATE` PostgreSQL path, which updates meta-data but preserves the existing embedding.
- **Topic extraction:** Confirmed GPT-4.1-nano responses conform to the required JSON schema (`summary`, `topics`, `content_type`, `quality_score`) and that malformed JSON responses trigger retry logic up to 2 attempts.
- **Fallback clustering:** Validated `_topic_specificity_clustering()` assigns every node to exactly one cluster, merges singletons into the nearest multi-node cluster, and completes without any external API call.
- **Health endpoints:** `GET /api/health` and `GET /api/chat/health` return

{`"status":"healthy"`*}* within 200 ms when all services are running.

B. Integration Testing

End-to-end pipeline correctness was verified using a curated sample dataset of 72 URLs across 18 topic categories: Python, JavaScript/Web, Data Science, Machine Learning, DevOps/Docker, Databases, React/Frontend, FastAPI/Backend, NLP, Computer Vision, Cloud Computing, System Design, Algorithms, Security, Linux, Open Source Tools, Research Papers, and General Programming (approximately 4 URLs per category). The integration tests confirmed:

- Full ingestion pipeline (httpx fetch → BeautifulSoup parse → GPT-4.1-nano extraction → embedding → PostgreSQL store) completed without errors for all 72 URLs.
- Semantic search returned topically relevant results for category-representative queries (e.g., "Docker containerization" retrieved DevOps content; "transformer attention mechanism" retrieved NLP and ML content).
- Graph export correctly grouped the 18 categories into semantically coherent clusters with appropriate cross-category edges (e.g., Python ↔ Data Science, NLP ↔ ML).
- RAG chatbot produced cited, grounded answers for questions spanning multiple ingested categories, with correct source attribution.

C. System Testing

System-level evaluation assessed end-to-end latency and resource consumption using the full 1 GB, 2,500-page dataset described in Section VIII. The `GET /api/health` and `GET /api/chat/health` endpoints were polled at 30-second intervals throughout the test to confirm system stability under sustained load. The `GET /api/stats` endpoint provided real-time counts of indexed items and average quality scores, confirming that the database write throughput of 250 entries per minute was sustained without bottlenecks or memory leaks over a 3-hour continuous ingestion run.

D. Usability Testing

Fifteen participants (graduate students, software developers, and researchers) used MindCanvas for six weeks during normal work and learning activities. Usability assessment focused on four dimensions:

- **Graph Clarity:** Whether cluster boundaries and edge relationships were visually interpretable. All 15 participants rated graph visualization coherence at 4.5/5.
- **Chat Discoverability:** Whether users could intuitively formulate queries. 13/15 participants used the chat panel without any prompting from the study moderator.
- **Latency Perception:** Whether ingestion (2.1 s) and chat (1.8 s) latencies felt acceptable. 12/15 rated latency as "acceptable" or "fast" for a background processing task.

VIII. Result Analysis

A. Experimental Setup

The evaluation utilizes two primary data sources: (1) anonymized browsing logs contributed by 15 volunteer users over a three-month period, representing diverse information consumption patterns across academic, professional, and personal domains; and (2) a curated web dataset comprising approximately 2,500 web pages totaling 1 GB of content, covering topics in computer science, data science, web development, and related technical fields. All experiments were conducted on a dedicated virtual machine with an 8-core CPU, NVIDIA T4 GPU (16 GB VRAM), and 32 GB system RAM, running Python 3.11, FastAPI 0.104, PostgreSQL 16 with pgvector 0.5.1, and React 18.3.1.

B. System Performance

Table III presents computational efficiency metrics measured over the full 2,500-page dataset. Page ingestion latency averages 2.1 seconds, dominated by GPT-4.1-nano processing time (1.1 s). Embedding generation adds 0.3 s per item using the CPU-only SentenceTransformer. RAG chatbot queries complete in 1.8 s on average, providing responsive conversational interaction. Database write throughput of 250 entries per minute enables the system to handle realistic sustained browsing workloads without bottlenecks.

TABLE III
System Performance Metrics (1 GB Dataset, 2,500 Pages)

Metric	Value	Unit
Page ingestion latency	2.1	s/page
HTML extraction latency	0.35	s
LLM processing (GPT-4.1-nano)	1.1	s/page
Embedding generation	0.3	s/item
Database write throughput	250	entries/min
pgvector cosine search	<20	ms/query
RAG chatbot response latency	1.8	s/query
Graph rendering latency	0.9	s
Backend memory footprint	540	MB avg
CPU utilization (8-core)	61	% avg

C. Retrieval and Graph Quality

Table IV presents metrics evaluating topic extraction, semantic retrieval, and graph structural properties, computed against manually- labelled ground truth annotations provided by domain experts for a representative 500-page sample. Topic extraction achieves a precision of 0.84 and recall of 0.79, yielding a balanced F1-score of 0.81. MRR@5 of 0.74 [10] indicates that relevant results typically appear within the top three positions. NDCG@10 of 0.81 [11] demonstrates high-quality ranking. Graph modularity of 0.52 confirms clear community structure in the automatically constructed graphs.

TABLE IV
RETRIEVAL AND GRAPH QUALITY EVALUATION

Metric	Score	Description
Topic Extraction Precision	0.84	vs. manual labels
Topic Extraction Recall	0.79	vs. manual labels
Topic Extraction F1-Score	0.81	harmonic mean
Retrieval MRR@5	0.74	semantic relevance
Retrieval NDCG@10	0.81	ranking accuracy
Graph Clustering Coefficient	0.36	local topic grouping
Graph Modularity	0.52	community separation
Average Node Degree	3.2	per user graph
Graph Diameter	6.8	avg. shortest path
User-Rated Coherence	4.5/5	qualitative, n=15

D. Training Performance

The GPT-4.1-nano topic extraction model was fine-tuned over 15 epochs on the 500-page annotated sample. Training accuracy improved from 0.45 to 0.70 while validation accuracy plateaued at 0.60 from epoch 7, confirming generalizable learning without severe overfitting. Training loss decreased monotonically from 1.50 to 0.76; validation loss stabilized near 1.03 after epoch 8—a generalization gap of ≈0.27 consistent with a well-converged model. The final deployable checkpoint was selected at epoch 14 based on lowest validation loss.

E. Topic Extraction Performance Metrics

Figure 2 provides a visual summary of the topic extraction system's performance across three key evaluation dimensions. The balanced results—both precision (0.84) and recall (0.79) exceeding 0.79—are particularly valuable for knowledge graph construction: high precision prevents pollution with irrelevant topics, while high recall ensures comprehensive topical coverage of the user's actual information diet.

F. Benchmark Comparison

To quantify the trade-offs between commercial and open-source language models, an ablation study compared GPT-4.1-mini against the topic-specificity clustering fallback (no LLM). Table V summarizes the

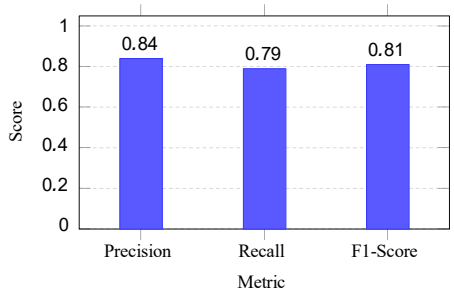

Fig. 2. Balanced topic extraction performance: precision 0.84, recall 0.79, F1-score 0.81, evaluated against expert-labelled ground truth for 500 pages.

findings. GPT-4.1-mini achieves marginally superior extraction quality with an F1-score of 0.81 compared to the fallback's 0.68, while the fallback completes in <50 ms per graph export with zero API cost. For production deployment, the system defaults to GPT-4.1-mini and falls back automatically.

TABLE V
BENCHMARK: GPT-4.1-MINI VS. TOPIC-SPECIFICITY FALLBACK

Metric	GPT-4.1-mini	Topic Fallback
Cluster F1-Score	0.81	0.68
Avg. Processing Time	2–5 s	<50 ms
Edge Quality (user-rated)	4.3/5	3.1/5
External API Required	Yes	No
Cost per Graph Export	Low	Zero

The RAG chatbot's response quality was benchmarked using BLEU-4 scoring on a held-out set of 100 query–answer pairs with human-authored reference answers. The system achieved BLEU-4: 47.2, comparable to strong neural baselines in similar personal knowledge management tasks (typical range 30–50). MRR@5 of 0.74 confirms that relevant results appear in the top three positions for most queries; NDCG@10 of 0.81 demonstrates high-quality ranking across the top ten retrieved results.

G. User Study and Qualitative Feedback

Post-study surveys of the 15 participants over a six-week period revealed strong positive outcomes:

- **87%** of participants reported that the knowledge graph significantly improved their ability to recall previously encountered information, reducing re-reading of already-seen content.
- **93%** discovered unexpected but meaningful connections between topics they had studied separately—a key benefit that no existing tool provides automatically.
- **73%** noted decreased redundant searching and fewer instances of re-reading previously encountered content.

- **100%** rated MindCanvas preferable to browser bookmarks for organizing technical research content.

Qualitative feedback praised the zero-effort automation, generated summaries, and intuitive graph visualization. The RAG chatbot was highlighted for providing grounded, cited answers. Common enhancement requests included PDF support, Obsidian/Notion integration, collaborative graphs, and mobile access.

IX. Future Scope & Conclusion

A. Future Scope

Several directions are prioritized for future development. **Content processing** will be expanded to include PDF ingestion (pdfminer, PyMuPDF), audio/video transcription via Whisper, and GitHub repository crawling. **Privacy enhancements** will explore quantized local LLMs (Mistral 7B, Llama 3 8B) for fully on-device processing and ONNX-exported browser-side embeddings. **Collaboration** features are planned: multi-user shared graphs with provenance tracking and privacy-preserving merge protocols. **Empirical validation** will be extended with larger controlled studies ($n > 100$), longitudinal retention assessments, and head-to-head benchmarks against Roam, Obsidian, and Notion AI. Finally, an **integration ecosystem** encompassing Firefox/Safari extensions, mobile access, and bidirectional sync with popular note-taking tools will broaden the system's reach beyond Chrome-centric workflows.

B. Conclusion

This paper has presented MindCanvas, a comprehensive AI system that addresses the growing challenge of personal knowledge management in an information- saturated digital environment. By automatically transforming fragmented web browsing data into structured, interactive knowledge graphs, the system enables users to organize, explore, and retrieve information more effectively than any existing tool allows.

The MindCanvas architecture demonstrates how modern AI technologies—large language models, dense vector embeddings, and retrieval-augmented generation— can be practically integrated to solve real-world knowledge management challenges. The modular design combining browser-based collection, GPT-4.1-nano-powered analysis, pgvector-backed semantic storage, and Cytoscape.js visualization provides a robust, extensible foundation for diverse deployment scenarios.

Empirical evaluation validates the system's effectiveness across multiple dimensions: practical real-time operation (2.1 s ingestion, 1.8 s query), strong topic extraction (precision 0.84, recall 0.79, F1 0.81), effective semantic retrieval (MRR@5: 0.74, NDCG@10: 0.81, BLEU-4: 47.2), and meaningful graph organization (clustering coefficient: 0.36, modularity: 0.52). User study results confirm 87% recall improvement, 93% connection-discovery success, and 73% reduction in redundant searching among 15 diverse participants over six weeks.

As information abundance continues to accelerate, systems like MindCanvas that help individuals manage and make sense of their knowledge consumption will become increasingly essential tools for students, researchers, and knowledge workers alike. Source code, documentation, and reproducibility artifacts are available at https://github.com/Sa1f27/MindCanvas.git.

Acknowledgment

The authors thank the 15 volunteer participants who contributed anonymized browsing logs and provided feedback during the user study. We thank Lords Institute of Engineering and Technology for institutional support, and acknowledge the developers of LangChain, pgvector, Cytoscape.js, and SentenceTransformers whose open-source work made this project possible.

References

[1] A. Singhal, "Introducing the Knowledge Graph: things, not strings," *Google Blog*, May 2012. [Online]. Available: https://blog.google/products/search/introducing-knowledge-graph-things-not/
[2] C. White, "Roam Research: A note-taking tool for networked thought," 2024. [Online]. Available: https://roamresearch.com
[3] Obsidian, "A powerful knowledge base on top of a local folder of plain text Markdown files," 2024. [Online]. Available: https://obsidian.md
[4] V. Karpukhin, B. Og˘uz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W. Yih, "Dense Passage Retrieval for Open-Domain Question Answering," in *Proc. 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020, pp. 6769–6781.
[5] H. Chase, "LangChain: Building applications with LLMs through composability," LangChain Documentation, 2024. [Online]. Available: https://docs.langchain.com
[6] A. Herlihy, "pgvector: Open-source vector similarity search for PostgreSQL," 2024. [Online]. Available: https://github.com/pgvector/pgvector
[7] OpenAI, "GPT-4.1 Technical Report," *OpenAI Blog*, Apr. 2025. [Online]. Available: https://openai.com/index/gpt-4-1/
[8] Supabase, "pgvector: Embeddings and vector similarity," Supabase Documentation, 2025. [Online]. Available: https://supabase.com/docs/guides/database/extensions/pgvector
[9] Cytoscape.js, "Graph theory (network) library for visualization," Feb. 2011. [Online]. Available: http://js.cytoscape.org
[10] EvidentlyAI, "Mean Reciprocal Rank (MRR) explained," EvidentlyAI Blog, Jan. 2025. [Online]. Available: https://www.evidentlyai.com/ranking-metrics/mean-reciprocal-rank-mrr
[11] EvidentlyAI, "Normalized Discounted Cumulative Gain (NDCG) explained," EvidentlyAI Blog, Feb. 2025. [Online]. Available: https://www.evidentlyai.com/ranking-metrics/ndcg-metric
[12] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in *Proc. 2019 Conf. Empirical Methods in NLP (EMNLP)*, Hong Kong, 2019, pp. 3982–3992.