PARSE

Presentation Analysis and Real-time Semantic Extraction

Final Report

Date: April 27, 2026

Team PARSE – T2501

22203783 Anıl Kılıç – DevOps and Infrastructure Lead
22202680 Aybars Buğra Aksoy – Backend Development Lead
22202981 Barış Yaycı – ML/AI Integration Lead
22203661 Bora Yetkin – Frontend Development Lead
22201772 Eren Berk Eraslan – Zoom and ML/AI Lead

Department of Computer Engineering
Bilkent University

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Purpose of the System

PARSE (Presentation Analysis and Real-time Semantic Extraction) is a multi-platform intelligent meeting assistant designed to revolutionize how users interact with and learn from presentations. The system addresses the growing need for accessible, AI-powered tools that can process real-time video and audio streams during meetings, providing instant transcription, slide analysis, and context-aware question answering.

1.1.1 Problem Statement

In today's digital age, remote meetings and presentations have become ubiquitous. However, participants face several challenges:

- **Information Overload**: Difficulty in capturing and retaining key information during lengthy presentations
- **Accessibility Barriers**: Hearing-impaired individuals lack real-time transcription
- **Engagement Gaps**: Participants who miss parts of a presentation cannot easily catch up
- **Post-Meeting Review**: No intelligent way to query presentation content after the fact
- **Multi-Platform Fragmentation**: Different meeting platforms (Zoom, LiveKit) require separate solutions

1.1.2 Solution Overview

PARSE provides a unified solution centered on a native desktop companion that sits beside Zoom, with the rest of the stack supporting it:

1. Ships a native macOS desktop companion (Electron + React) as the primary student surface — a quiet window beside Zoom that holds transcripts, Q&A, and the student's own notes.
2. Joins the student's Zoom meeting through a Recall.ai meeting bot rather than any in-Zoom embed, so the student keeps their normal Zoom workflow untouched.

3. Streams real-time speech-to-text via Deepgram Nova 3 (delivered through the Recall.ai bot) and broadcasts every line to the desktop companion over WebSocket.
4. Detects slide changes from the meeting share and processes each slide at the patch level with DocLayout-YOLO.
5. Runs per-patch OCR with GLM-OCR to extract Turkish text faithfully.
6. Embeds text patches with bge-m3 (1024-dim, multilingual) and non-text patches with mCLIP `xlm-roberta-base-ViT-B-32` (512-dim).
7. Correlates each spoken sentence with the slide that was on screen at the time so transcripts and slides remain linked at retrieval time.
8. Generates Turkish answers with Qwen3.5-4B grounded in the matched patches and the slide-correlated transcript window.

A self-hosted LiveKit room option is retained as a secondary path for environments that cannot use Zoom + Recall.ai, but it is not the primary surface and is documented as a legacy alternative throughout this report.

1.2 Design Goals

The following design goals guided the architectural decisions for PARSE:

Table 1.1: PARSE Design Goals and Priorities

Design Goal	Description	Priority
Usability	Intuitive interface requiring minimal setup; seamless integration with existing meeting workflows	High
Performance	Real-time processing with <3 second latency for transcription; <5 second response for Q&A	High
Reliability	99.5% uptime; graceful degradation when components fail	High
Scalability	Support concurrent meetings; horizontal scaling of processing nodes	Medium
Security	End-to-end encryption; secure authentication; GDPR compliance	High
Extensibility	Modular architecture allowing easy addition of new ML models and platforms	Medium
Maintainability	Clean separation of concerns; comprehensive logging; containerized deployment	Medium
Modularity	Independent microservices with well-defined APIs	High
Flexibility	Support for both cloud and on-premises deployment	Medium
Aesthetics	Modern, professional UI consistent with enterprise standards	Low

1.3 Definitions, Acronyms, and Abbreviations

Table 1.2: Definitions and Acronyms

Term	Definition
PARSE	Presentation Analysis and Real-time Semantic Extraction
LiveKit	Open-source, self-hosted WebRTC platform for real-time communication
RTMS	Real-Time Media Streams (Zoom's API for accessing meeting media)
WebRTC	Web Real-Time Communication protocol for peer-to-peer audio/video
STT	Speech-to-Text conversion
VLM	Vision Language Model (AI model that processes both images and text)
YOLO	You Only Look Once – real-time object detection algorithm
CLIP	Contrastive Language-Image Pre-training (OpenAI's vision-language model)
pgvector	PostgreSQL extension for vector similarity search
JWT	JSON Web Token for secure authentication
OAuth	Open Authorization protocol for secure delegated access
API	Application Programming Interface
CUDA	Compute Unified Device Architecture (NVIDIA GPU computing)

1.4 Overview

This Final Report is organized as follows:

- **Chapter 2**: Analyzes current software architecture including competitors and alternative solutions
- **Chapter 3**: Presents the proposed software architecture with detailed subsystem decomposition
- **Chapter 4**: Documents the services provided by each subsystem
- **Chapter 5**: Defines functional and non-functional test cases
- **Chapter 6**: Discusses engineering design considerations including constraints and standards
- **Chapter 7**: Details teamwork organization and collaboration
- **Chapter 8**: Provides a glossary of terms
- **Chapter 9**: Lists all references used

Chapter 2

Requirements Details

This chapter consolidates the refined functional and non-functional requirements that the implemented PARSE system was built to satisfy. Each requirement is given a stable identifier (FR-* / NFR-*), a priority, and the verification approach used in Chapter 8.

2.1 Functional Requirements

Table 2.1: Functional Requirements

ID	Description	Priority	Verified by
FR-AUTH-1	A new user can register with email + password and receive a session cookie via Better Auth.	Critical	TC-AUTH-001
FR-AUTH-2	A registered user can log in and resume an existing session.	Critical	TC-AUTH-002
FR-AUTH-3	Invalid credentials must not produce a session and must surface a non-revealing error.	Critical	TC-AUTH-003
FR-AUTH-4	A user can log out, invalidating the active session cookie.	Major	TC-AUTH-004
FR-DESK-2	A student can paste a Zoom meeting URL in the desktop companion and PARSE joins via a Recall.ai meeting bot, returning bot_id and meeting_id.	Critical	TC-DESK-002
FR-DESK-3	The desktop companion subscribes to the bot's public WebSocket and renders transcript lines as they arrive.	Critical	TC-DESK-003
FR-ROOM-1	A signed-in user can create a Zoom-integrated room from the dashboard.	Major	TC-ROOM-002

Table 2.1: Functional Requirements

ID	Description	Priority	Verified by
FR-ROOM-2	A user can leave / delete / list rooms through the dashboard.	Major	TC-ROOM-004, TC-ROOM-005, TC-ROOM-006
FR-ROOM-3	The legacy LiveKit subscriber bot stays hidden from the participant list.	Minor	TC-LK-001
FR-MEDIA-1	Real-time audio streams are forwarded to Deepgram Nova 3 (via Recall.ai in the primary path) and transcript lines are broadcast to clients.	Critical	TC-TRANS-001, TC-BOT-002
FR-MEDIA-2	Slide-share frames are captured for downstream processing — relayed by Recall.ai in the primary path, or read directly from Zoom RTMS / LiveKit screen tracks in the alternative paths.	Critical	TC-SLIDE-001, TC-ZOOM-002
FR-SLIDE-1	On detected slide change the bot uploads the slide to the GPU server's `/upload-and-process` endpoint within 3 s.	Critical	TC-SLIDE-001
FR-SLIDE-2	The GPU server segments each slide with DocLayout-YOLO and labels each patch with a DocStructBench class.	Critical	TC-SLIDE-002
FR-SLIDE-3	Text-bearing patches are OCR'd by GLM-OCR and the result is cached on disk so reprocessing the same slide does not re-run OCR.	Major	TC-SLIDE-003
FR-SLIDE-4	Text patches whose OCR contains $\geq$ 5 alphabetic characters are embedded with bge-m3 (1024-dim, fp16).	Major	TC-SLIDE-004
FR-SLIDE-5	Non-text patches are embedded with mCLIP `xlm-roberta-base-ViT-B-32` (512-dim, fp16).	Major	TC-SLIDE-005
FR-TRANS-1	The bot tags every transcript line with the slide that was on screen when it was spoken and persists the line via `POST /transcripts`.	Critical	TC-TRANS-002
FR-QA-1	A student can submit a Turkish question via `POST /ask` and receive a Turkish answer in $\leq$ 6 s on the reference 24 GB GPU.	Critical	TC-QA-001
FR-QA-2	The Q&A pipeline retrieves up to 5 unique top-text-matched slides (bge-m3) and 5 top non-text patches (mCLIP) and injects only those into Qwen3.5-4B.	Critical	TC-QA-002

Table 2.1: Functional Requirements

ID	Description	Priority	Verified by
FR-QA-3	Slide-correlated transcript lines for the union of retrieved slide IDs are added to the Qwen prompt as text context.	Major	TC-QA-003
FR-QA-4	When `regenerate=true` the most recent qa_history pair for the session is deleted before generation and sampling is enabled (temperature=0.9, top_p=0.92).	Major	TC-QA-004
FR-QA-5	When no patches, transcripts or notes exist for a meeting the server returns a friendly Turkish "no data" message rather than hallucinating.	Major	TC-QA-005
FR-LATEX-1	A natural-language math description sent to `POST /latex` returns a KaTeX-renderable LaTeX expression.	Minor	TC-LATEX-001
FR-RESET-1	`POST /reset` clears `slide_patches`, `transcripts`, and `qa_history` for a given meeting.	Major	TC-RESET-001
FR-DESK-1	The desktop companion can capture the active screen on macOS, draw on slides, and write LaTeX-rendered notes that thread into the same session.	Major	TC-DESK-001
FR-ZOOM-1	The Zoom sidebar app loads inside a Zoom meeting via the Meeting SDK and is invisible to other participants.	Major	TC-ZOOM-001

2.2 Non-Functional Requirements

Table 2.2: Non-Functional Requirements

ID	Description	Priority	Verified by
NFR-PERF-1	End-to-end Q&A latency on a single 24 GB GPU (bge-m3 fp16 + mCLIP fp16 + Qwen3.5-4B bf16, `enable_thinking=False`) must not exceed 6 s for a question with 5 patch images.	High	TC-PERF-001
NFR-PERF-2	GLM-OCR on a single text patch (avg. size $\sim$1024×256) must complete within 3 s warm.	High	TC-PERF-002
NFR-PERF-3	The transcript pipeline must surface a finalized line within 2 s of the speaker finishing.	High	TC-PERF-003
NFR-PERF-4	Slide-change detection cooldown $\leq$ 2 s; perceptual-hash threshold tuned for 95% same-slide stability.	Medium	TC-PERF-004

Table 2.2: Non-Functional Requirements

ID	Description	Priority	Verified by
NFR-VRAM-1	The full GPU server must fit within 22 GB of VRAM during `generate()` on the reference hardware.	High	TC-PERF-005
NFR-SEC-1	Authentication must use httpOnly session cookies; the proxy secret must guard the backend.	High	TC-SEC-001
NFR-SEC-2	Slide and transcript data must be isolated by `meeting_id` — queries must never return another meeting's rows.	High	TC-SEC-002
NFR-SEC-3	No user-supplied prompt or transcript may bypass the system prompt's grounding rules.	High	TC-SEC-003
NFR-LANG-1	The default Q&A surface must be Turkish; the system prompt must enforce Turkish-only output.	High	TC-USAB-001
NFR-REL-1	The GPU server must restart cleanly without manual schema migration; `INIT_SQL` is idempotent across versions.	Medium	TC-REL-001
NFR-REL-2	A failed transcript-store call must log a warning but never block downstream broadcast.	Medium	TC-REL-002
NFR-COMPAT-1	The full stack must run on Linux x86_64 hosts with Docker 24+ and on Apple Silicon (`platform: linux/amd64` bot fallback).	Medium	TC-COMPAT-001
NFR-USAB-1	New users must complete first-run setup (signup, room create, sidebar load) within 3 minutes without consulting a manual.	Medium	TC-USAB-002
NFR-MAINT-1	New ML model paths can be swapped through a single `PROJECT_ROOT` environment variable.	Medium	TC-MAINT-001

2.3 Requirements Traceability Summary

Every functional requirement above is bound to at least one test case in Chapter 8. The non-functional requirements feed both the functional/non-functional split inside that chapter and the maintenance plan in Chapter 9, where measurement strategy and SLAs for the same metrics are defined.

Chapter 3

Current Software Architecture

3.1 Market Analysis

The meeting assistant market has grown significantly, with several competitors offering various features. This section analyzes existing solutions to position PARSE's unique value proposition.

3.2 Competitor Analysis

3.2.1 Otter.ai

Overview: Cloud-based AI meeting assistant focusing on transcription and note-taking.

Features:

- Real-time transcription
- Speaker identification
- Meeting summaries
- Integration with Zoom, Google Meet, Microsoft Teams

Limitations:

- No slide analysis capability
- No context-aware Q&A
- Cloud-only (no self-hosted option)
- Subscription-based pricing

3.2.2 Microsoft Copilot in Teams

Overview: AI assistant integrated into Microsoft Teams.

Features:

- Meeting transcription
- Action item extraction

- Post-meeting summaries
- Integration with Microsoft 365

Limitations:

- Limited to Microsoft ecosystem
- No real-time slide analysis
- Enterprise-only pricing
- No open-source option

3.2.3 Zoom AI Companion

Overview: Native AI features within Zoom platform.

Features:

- Meeting summaries
- Smart recording highlights
- Chat compose assistance

Limitations:

- Limited to Zoom platform
- No slide content analysis
- No semantic Q&A capability
- Requires Zoom paid plans

3.3 Comparative Analysis

Table 3.1: Feature Comparison Matrix

Feature	PARSE	Otter.ai	MS Copilot	Zoom AI
Real-time Transcription	✓	✓	✓	✓
Slide Detection	✓	×	×	×
Slide Content Analysis	✓	×	×	×
Context-aware Q&A	✓	×	Partial	×
Multi-platform Support	✓	✓	×	×
Self-hosted Option	✓	×	×	×
Open Source	✓	×	×	×
Vector Semantic Search	✓	×	×	×

3.4 PARSE Differentiators

PARSE's unique value proposition includes:

1. **Patch-Level Slide Processing**: DocLayout-YOLO segments each slide into typed regions; GLM-OCR extracts text per patch; bge-m3 and mCLIP embed text and non-text patches into pgvector for hybrid retrieval
2. **Vision Language Model Q&A**: Qwen3.5-4B answers in Turkish, grounded in the matched patches and the slide-correlated transcript window
3. **Multi-Platform Architecture**: Single backend supports both LiveKit (self-hosted) and Zoom (enterprise)
4. **Privacy-First Design**: Self-hosted option keeps all data on-premises
5. **Vector-Based Retrieval**: pgvector enables semantic search across all processed slides

Chapter 4

Proposed Software Architecture

4.1 Overview

PARSE follows a microservices architecture with clear separation of concerns. The student-facing surface is a native desktop companion that talks directly to a small set of services over REST and WebSocket.

4.1.1 Architectural Style

The system employs a **layered microservices architecture** with the following characteristics:

- **Presentation Layer**: Native desktop companion (Electron + React, primary surface) plus a Next.js account dashboard (sign-in / management) and an optional in-Zoom sidebar
- **API Gateway Layer**: FastAPI backend handling authentication and account management
- **Orchestration Layer**: Bot service that drives Recall.ai meeting bots, fans transcripts to clients, and forwards slides to the GPU server
- **Intelligence Layer**: GPU server running the patch-level ML pipeline
- **Data Layer**: PostgreSQL with pgvector extension

4.1.2 System Architecture Diagram

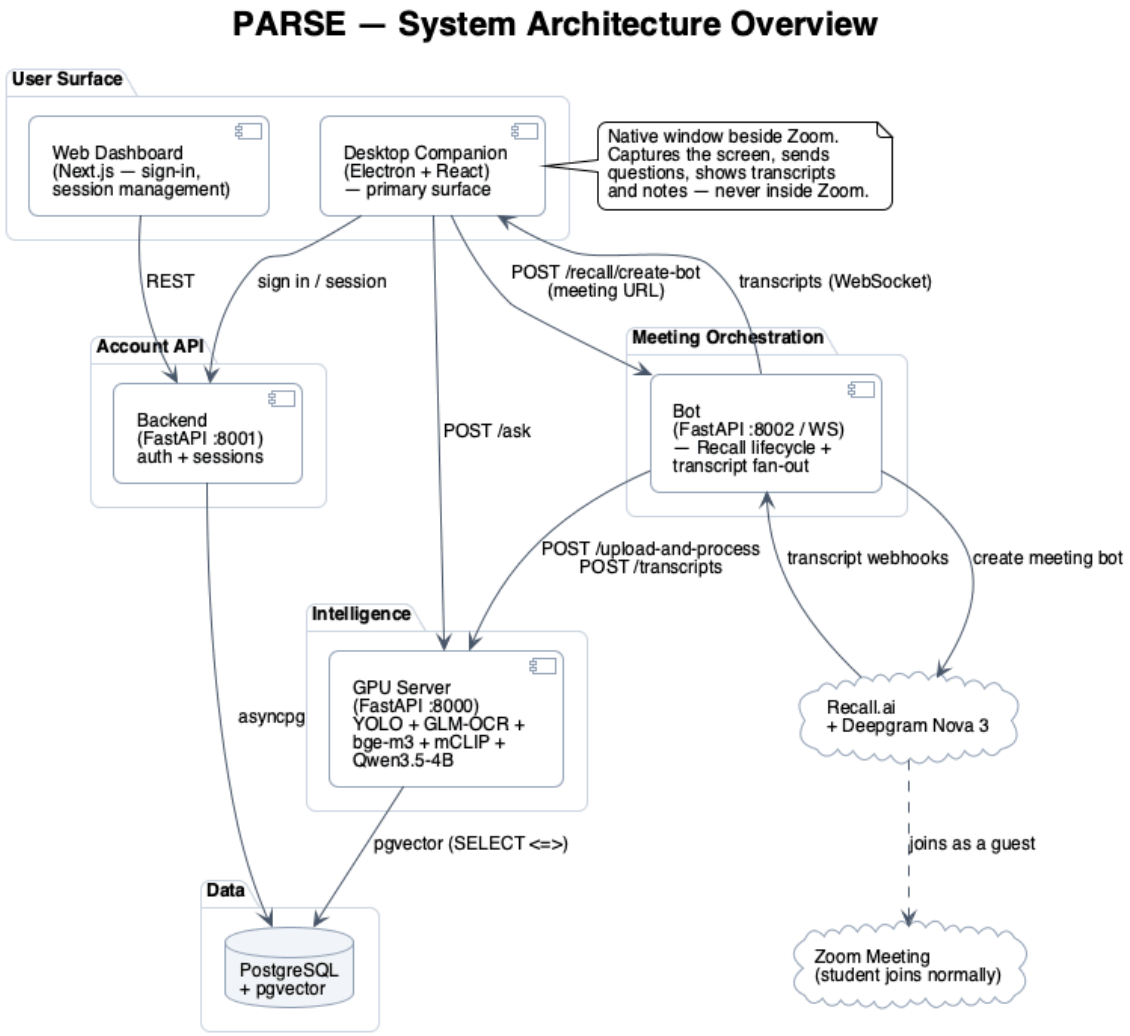

Figure 4.1: PARSE System Architecture Overview

4.2 Subsystem Decomposition

The PARSE system is decomposed into the following subsystems:

4.2.1 Desktop Companion Subsystem (Electron + React) — primary user surface

Purpose: The primary surface students interact with during a lecture. A native macOS window that sits beside Zoom and holds the live transcript, the Q&A panel, the notes editor, and the math/drawing tools.

Technology Stack:

- Electron 29 (`BrowserWindow` with `hiddenInset` title bar, `vibrancy: under-window`)
- React 18, react-scripts

- BlockNote (block-based editor) for the notes panel
- KaTeX for inline math (rendered through `rehype-katex` + `remark-math`)
- `react-markdown` for assistant answer rendering

Key Components:

- `App Shell`: session control (paste meeting URL, start/stop, persist across reload via `localStorage`), WebSocket client to the bot.
- `TranscriptionPanel`: live transcript fed by the bot's public WebSocket.
- `QAPanel`: question composer with Edit / Regenerate / Copy actions; the Regenerate path sets `regenerate=true` on the GPU server.
- `NotesPanel`: BlockNote editor with embedded `DrawingBlock` (sketches), `MathBlock`, and a `LatexAssistant` (calls `/latex`). Notes can be promoted into the next Q&A as `notes_context`.
- `ProfilePanel`: signed-in user, sign-out, link to the web dashboard.
- Electron `main.js`: 580×620 window, dev server on `:3002`, packaged build loads from `build/index.html`.

The desktop companion talks directly to the Bot (`POST /recall/create-bot`, WebSocket transcripts) and to the GPU server (`POST /ask`, `POST /latex`). It uses the Backend only for sign-in.

4.2.2 Web Dashboard Subsystem (Next.js) — account management

Purpose: Account-management surface for sign-up, sign-in, and (in the legacy path) creating self-hosted LiveKit rooms. It is not the primary place students live during a lecture — the desktop companion is. It exists so accounts can be managed in any browser and so the LiveKit-room option is still available for environments without Zoom + Recall.ai.

Technology Stack:

- Next.js 16 with App Router, React 19
- Better Auth for authentication (httpOnly session cookies)
- TailwindCSS v4
- LiveKit React SDK (legacy path only)

Key Components:

- `Auth Pages`: login and signup forms.
- `Dashboard`: account view; lists previous sessions and (legacy path) self-hosted LiveKit rooms.
- `API Proxy`: session-validating proxy to the backend.

4.2.3 Backend Subsystem (FastAPI)

Purpose: Handles authentication, room management, and orchestrates communication between services.

Technology Stack:

- FastAPI (Python async framework)
- asyncpg for database access
- httpx for async HTTP client
- LiveKit Python SDK

Key Services:

- `LiveKitService`: JWT token generation for room access
- `BotService`: Bot notification and coordination
- `ZoomService`: Zoom OAuth and SDK signature generation
- `RoomStore`: In-memory room state management
- `AuthService`: Session validation with Better Auth

4.2.4 Bot Subsystem

Purpose: Orchestrates Recall.ai meeting bots, fans transcripts out to the desktop companion, and forwards slides + transcript lines to the GPU server. The primary integration is Recall.ai; a LiveKit subscriber path is retained for the legacy self-hosted room option.

Technology Stack:

- FastAPI (port 8002 for both the REST API and the public WebSocket fan-out)
- Recall.ai HTTP client (bot lifecycle and transcript webhooks)
- Deepgram Nova 3 (transcripts arrive through Recall.ai)
- LiveKit Python SDK (legacy subscriber path, optional)
- PIL for slide-frame handling

Key Handlers:

- `RecallHandler`: `POST /recall/create-bot` (called by the desktop companion when the user pastes a Zoom URL), `POST /recall/webhook` (transcript and bot-status events from Recall.ai), `DELETE /recall/bot/{id}` (clean stop). This is the primary path.
- `TranscriptForwarder`: takes each finalized transcript line, tags it with the current `slide_id`, posts it to the GPU server's `/transcripts`, and pushes it to the public WebSocket so the desktop companion can render it live.

- `SlideDetector`: perceptual-hash slide-change detector running over deskshare frames; on change, the current frame is forwarded to `POST /upload-and-process` on the GPU server.
- `LiveKitSubscriber` (legacy): subscribes (hidden) to audio + screen-share tracks for self-hosted LiveKit rooms when those are used instead of Recall.ai.

4.2.5 GPU Server Subsystem

Purpose: Runs the patch-level ML pipeline for slide ingest, transcript correlation, and Turkish question answering.

Technology Stack:

- FastAPI with `asyncpg` and `pgvector` for persistent per-patch storage
- PyTorch with CUDA, fp16/bf16 mixed precision (24 GB VRAM budget)
- DocLayout-YOLO (`doclayout_yolo_docstructbench_imgsz1024`) for region detection + DocStructBench class labels
- GLM-OCR (`zai-org/GLM-OCR`) for per-patch Turkish-capable OCR
- bge-m3 (`BAAI/bge-m3`, fp16, 1024-dim) for multilingual text embeddings
- mCLIP `xlm-roberta-base-ViT-B-32` (fp16, 512-dim) for multilingual image embeddings on non-text patches
- Qwen3.5-4B (`Qwen/Qwen3.5-4B`, bf16) loaded through `AutoModelForImageTextToText`

Key Components:

- `Slide Ingest`: YOLO patches a slide, classifies each patch as text-bearing or non-text using DocStructBench labels, runs GLM-OCR on text patches with disk-cached output, then routes results to the two embedders.
- `Embedders`: bge-m3 over OCR text (with a 5-letter noise filter) and mCLIP over non-text patch crops — one row per patch in `slide_patches` with two nullable embedding columns.
- `Retrieval`: two independent pgvector queries per question — bge-m3 for text patches deduplicated to top-5 unique slides, mCLIP for top-5 non-text patches.
- `Generator`: Qwen3.5-4B receives the patch crops as images, the chosen slides' full OCR as text, the slide-correlated transcript window, and the user question; `enable_thinking=False` keeps responses lean.
- `Transcripts Endpoint`: stores `(meeting_id, slide_id, speaker, text, timestamp_ms)` rows so speech and slides remain linked at query time.

4.2.6 Zoom App Subsystem (legacy / optional)

Purpose: A small React application that can be loaded inside Zoom's own sidebar via the Zoom Meeting SDK, mirroring a subset of the desktop companion's panels (transcript + Q&A) for users who prefer to stay inside Zoom rather than open the standalone window. The desktop companion is the primary surface; this subsystem is retained as an optional in-Zoom alternative.

Technology Stack:

- React 18
- Zoom Meeting SDK
- WebSocket client (same fan-out as the desktop companion)

4.2.7 Data Subsystem

Purpose: Persistent storage for user sessions, slide embeddings, and Q&A history.

Technology Stack:

- PostgreSQL 15
- pgvector extension for vector similarity search
- Better Auth schema for user management

4.2.8 Component Diagram

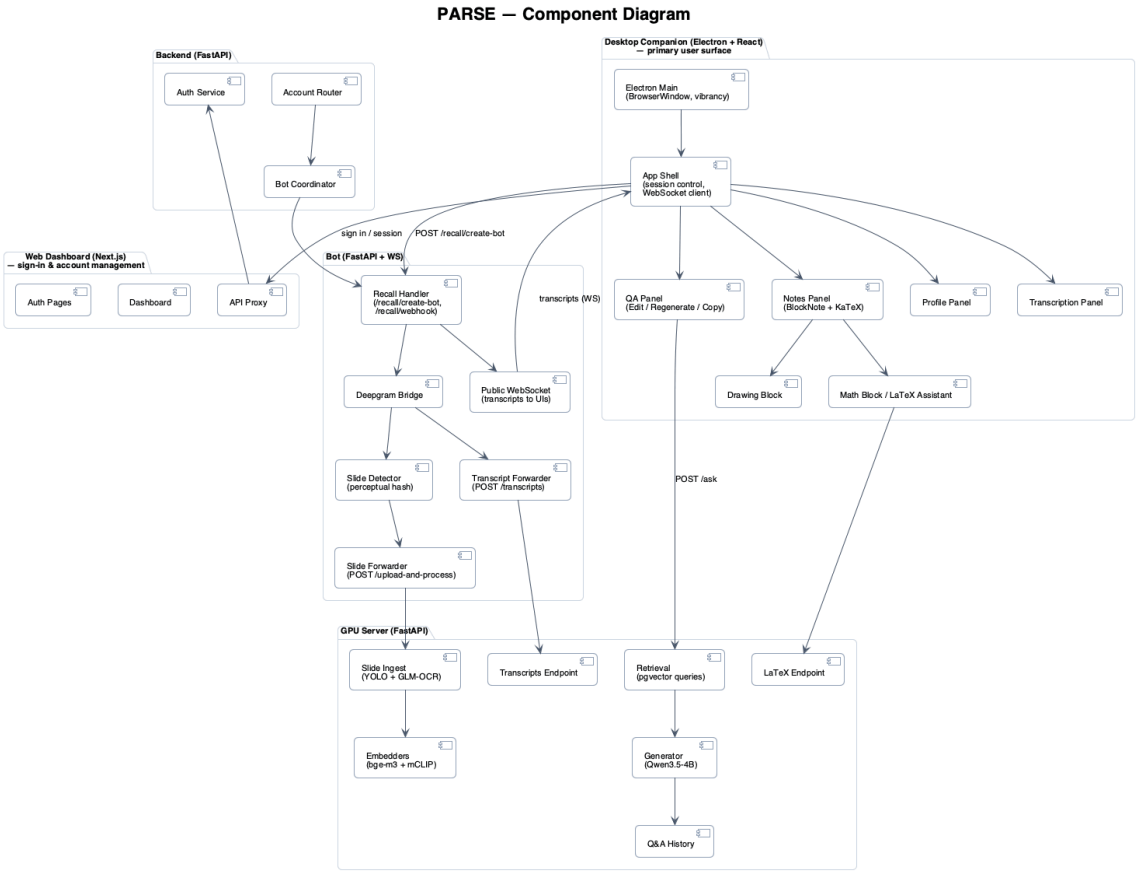

Figure 4.2: PARSE Component Diagram

4.3 Hardware/Software Mapping

4.3.1 Deployment Architecture

PARSE is designed for deployment on a single server with GPU capability, using Docker Compose for containerization.

Table 4.1: Hardware Requirements

Component	Minimum	Recommended
CPU	8 cores	16 cores
RAM	16 GB	32 GB
GPU	NVIDIA T4 (16GB)	NVIDIA RTX 4090 (24GB)
Storage	100 GB SSD	500 GB NVMe SSD
Network	100 Mbps	1 Gbps

4.3.2 Port Allocation

Table 4.2: Service Port Allocation

Port	Service	Purpose
3000	Frontend	Next.js account dashboard
3001	Zoom App	React in-Zoom sidebar (optional)
3002	Desktop Companion (dev)	React dev server consumed by Electron
5432	PostgreSQL	Database with pgvector
8000	GPU Server	ML inference (direct, not Docker)
8001	Backend	FastAPI REST API (auth, sessions)
8002	Bot	Bot REST API + public WebSocket (Recall.ai integration)
8080	Bot RTMS webhook	Zoom RTMS webhook server (alternative path)
6379	Redis	LiveKit state storage (legacy path only)
7880	LiveKit	WebRTC signaling (legacy path only)
7881–7882	LiveKit	UDP media ports (legacy path only)

4.3.3 Deployment Diagram

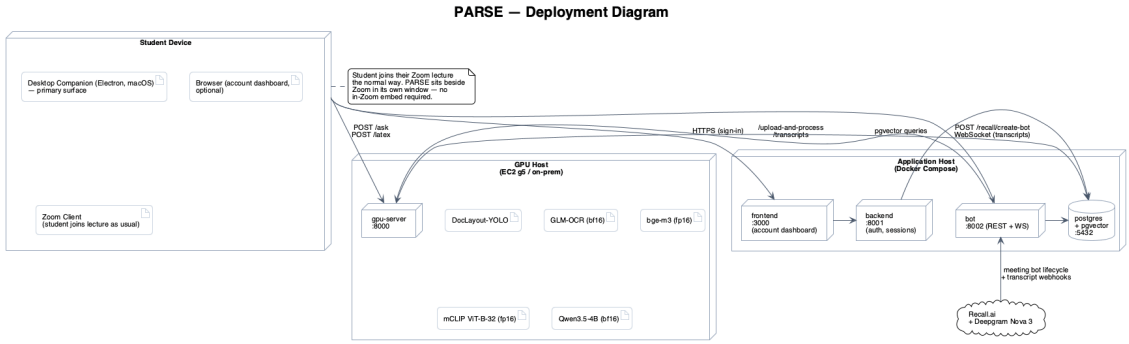

Figure 4.3: PARSE Deployment Architecture

4.4 Persistent Data Management

4.4.1 Database Schema

PARSE uses PostgreSQL with the pgvector extension for efficient vector similarity search. The schema is divided into two logical groups:

Authentication Tables (Better Auth)

- `user`: User accounts with email, name, and metadata
- `session`: Active session tokens with expiry
- `account`: OAuth provider tokens (Zoom integration)

- `verification`: Email verification codes

ML/Analytics Tables

The patch-level pipeline stores one row per YOLO patch in `slide_patches`, with two nullable embedding columns: `emb_bge` (1024-dim, populated only for text-bearing patches whose OCR passes the noise filter) and `emb_mclip` (512-dim, populated only for non-text patches). The `transcripts` table preserves the slide-speech link for retrieval-time correlation.

```
CREATE TABLE slide_patches (
    id           UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    meeting_id   VARCHAR(255) NOT NULL,
    slide_id     VARCHAR(255) NOT NULL,
    patch_id     INT NOT NULL,
    class_label  VARCHAR(100) NOT NULL,
    is_text      BOOLEAN NOT NULL,
    bbox         INT[],                  -- [x1, y1, x2, y2]
    image_path   VARCHAR(500),
    ocr_text     TEXT,                   -- text patches only
    emb_bge      vector(1024),           -- bge-m3, text patches only
    emb_mclip    vector(512),            -- mCLIP, non-text patches only
    created_at   TIMESTAMPTZ DEFAULT NOW()
);
CREATE INDEX idx_patches_meeting ON slide_patches(meeting_id);
```

Listing 4.1: Slide Patches Table (patch-level pipeline)

```
CREATE TABLE transcripts (
    id           UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    meeting_id   VARCHAR(255) NOT NULL,
    slide_id     VARCHAR(255),
    speaker      VARCHAR(255) NOT NULL,
    text         TEXT NOT NULL,
    is_final     BOOLEAN DEFAULT TRUE,
    confidence   REAL DEFAULT 1.0,
    source       VARCHAR(50),
    timestamp_ms BIGINT NOT NULL,
    created_at   TIMESTAMPTZ DEFAULT NOW()
);
CREATE INDEX idx_transcripts_meeting ON transcripts(meeting_id);
CREATE INDEX idx_transcripts_slide   ON transcripts(meeting_id,
   slide_id);
```

Listing 4.2: Transcripts Table (slide-speech correlation)

```
CREATE TABLE qa_history (
    id          UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    session_id  VARCHAR(255) NOT NULL,
    meeting_id  VARCHAR(255) NOT NULL,
    question    TEXT NOT NULL,
    answer      TEXT NOT NULL,
    created_at  TIMESTAMPTZ DEFAULT NOW()
);
CREATE INDEX idx_qa_session ON qa_history(session_id);
```

Listing 4.3: Q&A History Table Schema

4.4.2 Entity-Relationship Diagram

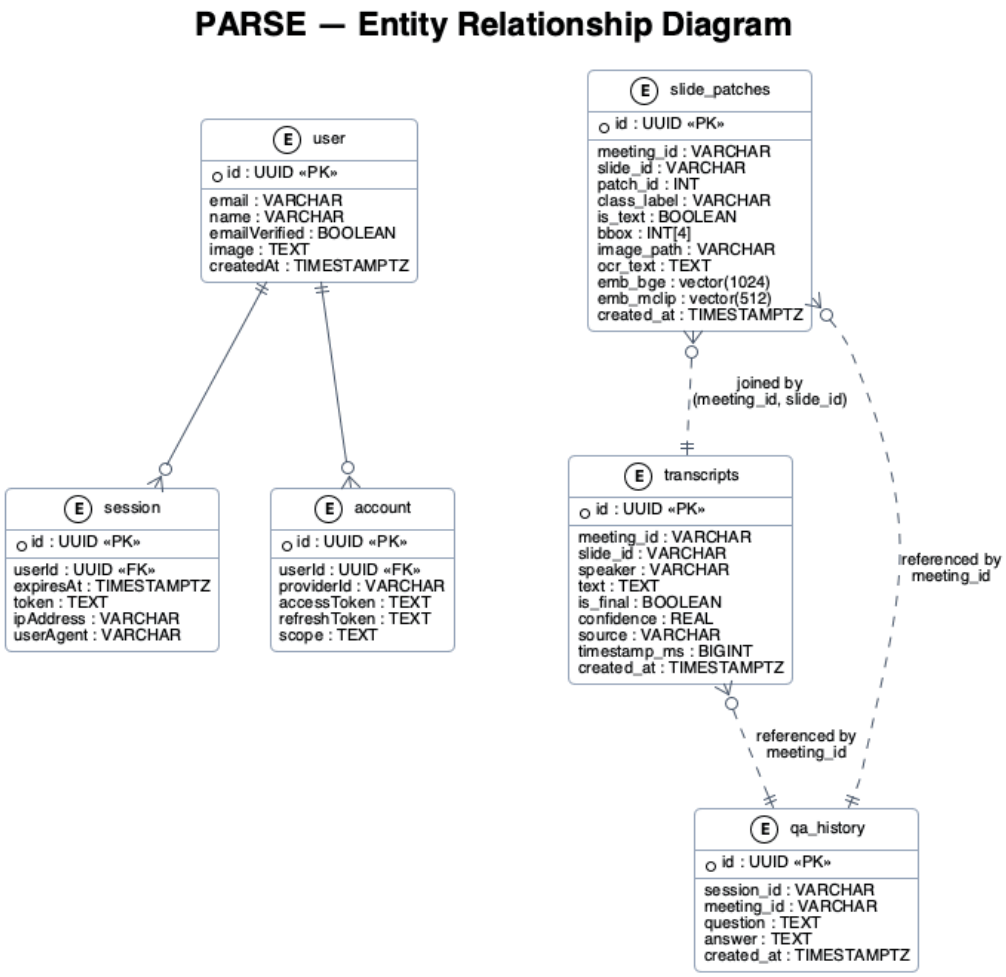

Figure 4.4: PARSE Database Entity-Relationship Diagram

4.5 Access Control and Security

4.5.1 Authentication Flow

PARSE implements a multi-layered authentication system:

1. **User Authentication**: Better Auth manages user sessions with httpOnly cookies
2. **API Proxy Security**: Next.js API routes validate sessions before forwarding to backend
3. **Backend Validation**: FastAPI middleware verifies proxy secret and user headers
4. **LiveKit Tokens**: JWT tokens with specific video grants per user role

4.5.2 Security Measures

Table 4.3: Security Implementation Details

Security Aspect	**Implementation**
Session Management	Better Auth with PostgreSQL-backed sessions; httpOnly cookies prevent XSS
API Security	Proxy secret header (`X-Proxy-Secret`) validates internal requests
Token Security	LiveKit JWTs expire after 6 hours; include specific room permissions
OAuth Security	Zoom OAuth tokens stored server-side; refresh before 5-minute expiry
Transport Security	All external communication over HTTPS/WSS
Data Isolation	Meeting data isolated by `meeting_id`; no cross-meeting access

4.5.3 LiveKit Token Grants (legacy path)

The primary join path goes through Recall.ai and does not require LiveKit. For the optional self-hosted LiveKit room flow, the backend mints two distinct JWT grants per room: a user grant with full publish/subscribe capability and a bot grant restricted to subscription with the `hidden` flag set so the bot does not appear in the participant list.

```
# User Token (can publish video/audio, subscribe to all)
user_grants = VideoGrants(
    room=room_name,
    room_join=True,
    can_publish=True,
    can_subscribe=True,
    can_publish_data=True
)

# Bot Token (subscribe only, publish transcription data)
bot_grants = VideoGrants(
    room=room_name,
    room_join=True,
    can_publish=False,      # Cannot send video/audio
    can_subscribe=True,     # Can receive all tracks
    can_publish_data=True,  # Can send transcriptions
    hidden=True             # Not visible in participant list
)
```

Listing 4.4: Token Generation with Video Grants (legacy LiveKit path)

Chapter 5

Subsystem Services

This chapter documents the API contracts and services provided by each subsystem.

5.1 Backend API Services

5.1.1 Room Management API

Table 5.1: Room Management Endpoints

Method	Endpoint	Description
POST	/api/rooms	Create a new room
GET	/api/rooms	List all rooms for current user
GET	/api/rooms/{room_id}	Get room details
POST	/api/rooms/{room_id}/join	Join room and get access token
POST	/api/rooms/{room_id}/end	End the room session
DELETE	/api/rooms/{room_id}	Delete a room

Create Room Request/Response

```
# Request
POST /api/rooms
{
    "name": "Team Meeting",
    "platform": "livekit"  # or "zoom"
}

# Response (201 Created)
{
    "id": "room_abc123",
    "name": "Team Meeting",
    "platform": "livekit",
    "creator_id": "user_xyz",
    "created_at": "2024-01-15T10:30:00Z",
    "livekit_room_name": "parse-room_abc123"
}
```

Listing 5.1: Create Room API Contract

Join Room Request/Response

```
# Request
POST /api/rooms/{room_id}/join

# Response (200 OK)
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "ws_url": "wss://livekit.example.com:7880"
}
```

Listing 5.2: Join Room API Contract

5.1.2 Zoom Integration API

Table 5.2: Zoom Integration Endpoints

Method	Endpoint	Description
GET	/api/zoom/authorize	Initiate Zoom OAuth flow
GET	/api/zoom/callback	Handle OAuth callback
GET	/api/zoom/token/{user_id}	Get user's access token
GET	/api/zoom/status/{user_id}	Check authorization status
POST	/api/zoom/refresh/{user_id}	Refresh access token
DELETE	/api/zoom/revoke/{user_id}	Revoke authorization

5.2 Bot API Services

5.2.1 Recall.ai Integration (primary)

The desktop companion drives the bot through Recall.ai endpoints. Each session is tied to a single bot ID returned at creation time; transcripts arrive via webhook and are fanned out over the public WebSocket.

Table 5.3: Bot Recall.ai Endpoints (primary)

Method	Endpoint	Description
POST	/recall/create-bot	Create a Recall.ai meeting bot for the supplied Zoom URL; returns {bot_id, meeting_id}
POST	/recall/webhook	Receive transcript and bot-status events from Recall.ai
DELETE	/recall/bot/{bot_id}	Stop and clean up an active bot
WS	/ws/{meeting_id}	Public transcript fan-out to the desktop companion
GET	/status	Bot service health

5.2.2 LiveKit Integration (legacy, optional)

Retained so self-hosted LiveKit rooms can still be used in environments without Zoom + Recall.ai. The desktop companion does not consume this path; the legacy web dashboard

does.

Table 5.4: Bot LiveKit Endpoints (legacy)

Method	Endpoint	Description
POST	/join-room	Bot joins a LiveKit room
DELETE	/leave-room/{room_name}	Bot leaves a room

5.2.3 Zoom RTMS Integration (alternative)

PARSE also supports a direct Zoom RTMS integration in which the bot subscribes to Zoom's Real-Time Media Streams (deskshare and transcript callbacks) instead of going through Recall.ai. Recall.ai is the default because it bundles the Zoom join lifecycle, OAuth, and Deepgram bridging into a single hosted service, which is faster to operate; the RTMS path remains available for deployments that prefer a direct Zoom integration without a third-party broker.

Table 5.5: Bot Zoom RTMS Endpoints (alternative path)

Method	Endpoint	Description
POST	/zoom/join	Register meeting for RTMS
DELETE	/zoom/leave/{meeting_id}	Stop RTMS processing

5.3 GPU Server API Services

5.3.1 Slide Ingest

```
# Request
POST /upload-and-process
Content-Type: multipart/form-data
- meeting_id: "zoom-meeting-123"
- slide_id:   "turkce-008"          # optional; auto-generated if absent
- image:      <slide_image.png>
- text_file:  <unused>               # accepted for backward-compatibility

# Server-side flow
#   1. DocLayout-YOLO -> patches with DocStructBench class_label
#   2. is_text = class_label in TEXT_CLASSES
#   3. Text patches      -> GLM-OCR (disk-cached) -> bge-m3 fp16
#   4. Non-text patches  -> mCLIP fp16
#   5. INSERT one row per patch into slide_patches

# Response (200 OK)
{
  "slide_id":   "turkce-008",
  "meeting_id": "zoom-meeting-123",
  "text_chunks": 6,    # is_text patches
  "patches":    11     # total patches
}
```

Listing 5.3: Patch-level ingest endpoint

5.3.2 Transcript Ingest

```
POST /transcripts
{
  "meeting_id": "zoom-meeting-123",
  "lines": [
    {
      "slide_id":     "turkce-008",     # current slide when spoken
      "speaker":      "Lecturer",
      "text":         "Bu slaytta ulnu harfi dusmus kelimeler var.",
      "is_final":     true,
      "confidence":   0.96,
      "source":       "deepgram-nova3",
      "timestamp_ms": 1714086500123
    }
  ]
}
```

Listing 5.4: Slide-correlated transcript ingest endpoint

5.3.3 Question Answering

```
# Request
POST /ask
{
  "question":           "11. sorunun cevabi",
  "meeting_id":         "zoom-meeting-123",
  "top_k":              5,
  "session_id":         "550e8400-...-446655440000",  # optional
  "transcript_context": null,     # optional override
  "notes_context":      null,
  "notes_images":       [],
  "regenerate":         false      # true on Edit/Regenerate from UI
}

# Server-side flow
#   1. bge-m3 ranks patches with emb_bge -> top-5 unique slides
#   2. mCLIP ranks patches with emb_mclip -> top-5 visual patches
#   3. Pull transcripts WHERE slide_id IN (...) for slide-correlated
   speech
#   4. Build Qwen3.5-4B payload: 5 patch crops as images
#                              + concatenated slide OCR as text
#                              + slide-correlated transcripts as text
#                              + Turkish system prompt
#                              + apply_chat_template(enable_thinking=
   False)
#   5. INSERT new pair into qa_history (drop the previous one when
   regenerate=true)

# Response (200 OK)
{
  "question": "11. sorunun cevabi",
  "answer":   "Dogru sik C'dir. Slayt OCR'inde verilen secenekler
   arasinda 'sariyim' ifadesi 'sari' kokune iyelik eki getirilerek
   olusturulmus tek ornektir.",
  "meeting_id": "zoom-meeting-123",
  "session_id": "550e8400-...-446655440000",
```

```
  "sources": {
    "image_patches": [{"slide_id": "turkce-008", "path": ".../patch_3.
    png"}],
    "text_snippets": [{"slide_id": "turkce-008", "text": "[slide=turkce
    -008 ...] 11. ..."}]
  }
}
```

Listing 5.5: Patch-level retrieval + Qwen3.5-4B answer

5.4 Sequence Diagrams

5.4.1 Room Creation and Join Flow

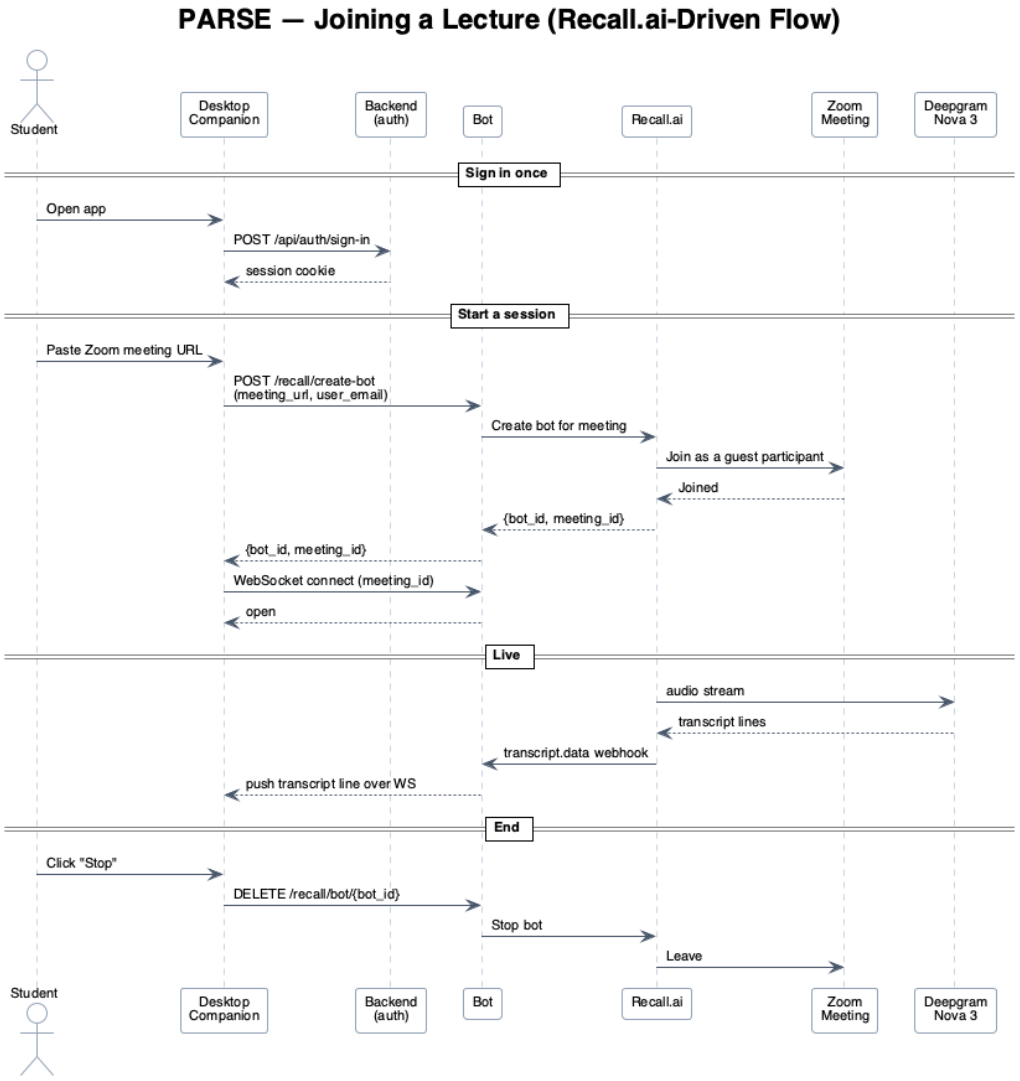

Figure 5.1: Room Creation and Join Sequence

5.4.2 Slide Processing and Q&A Flow

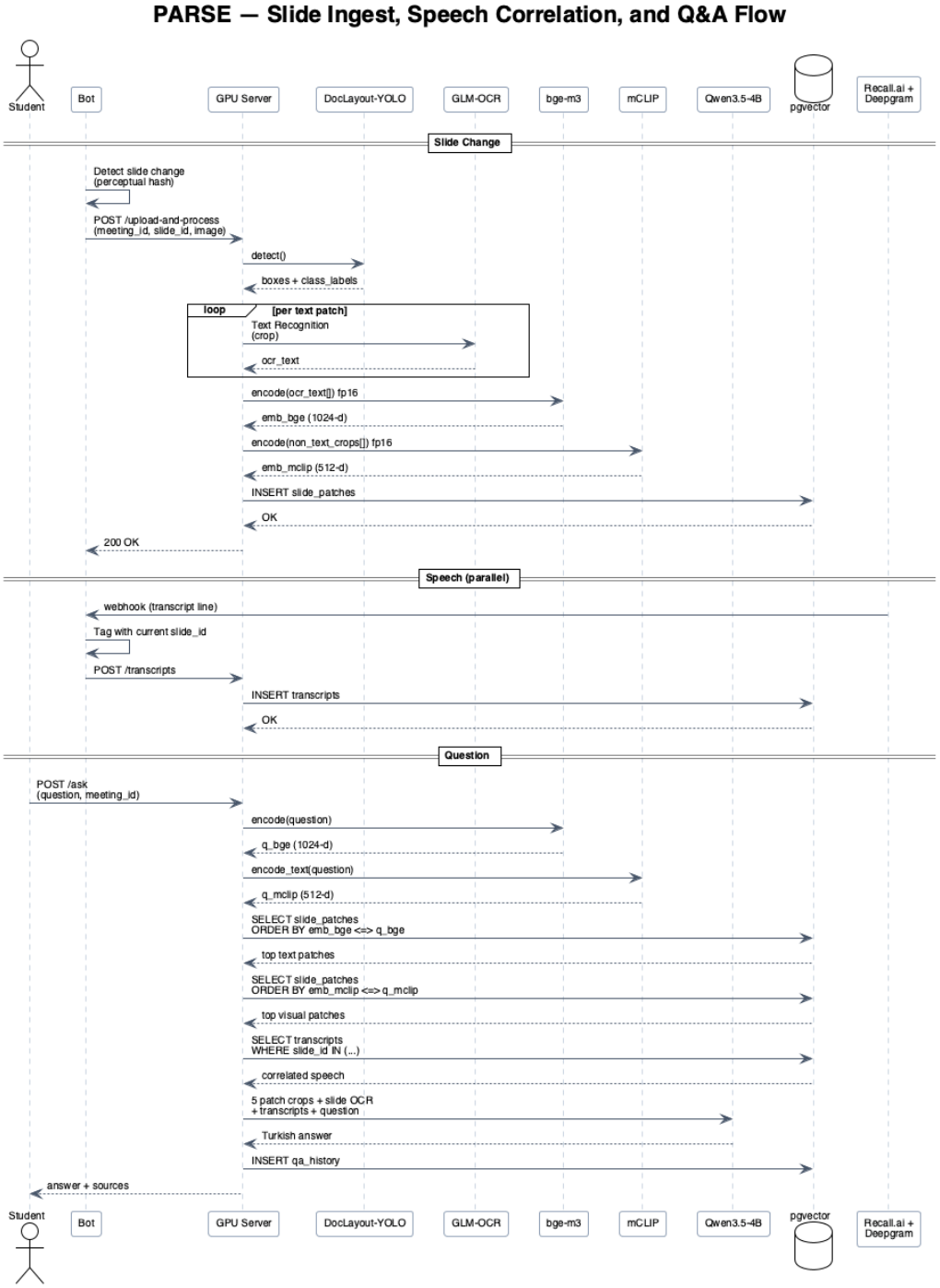

Figure 5.2: Slide Processing and Q&A Sequence

5.5 Class Diagram

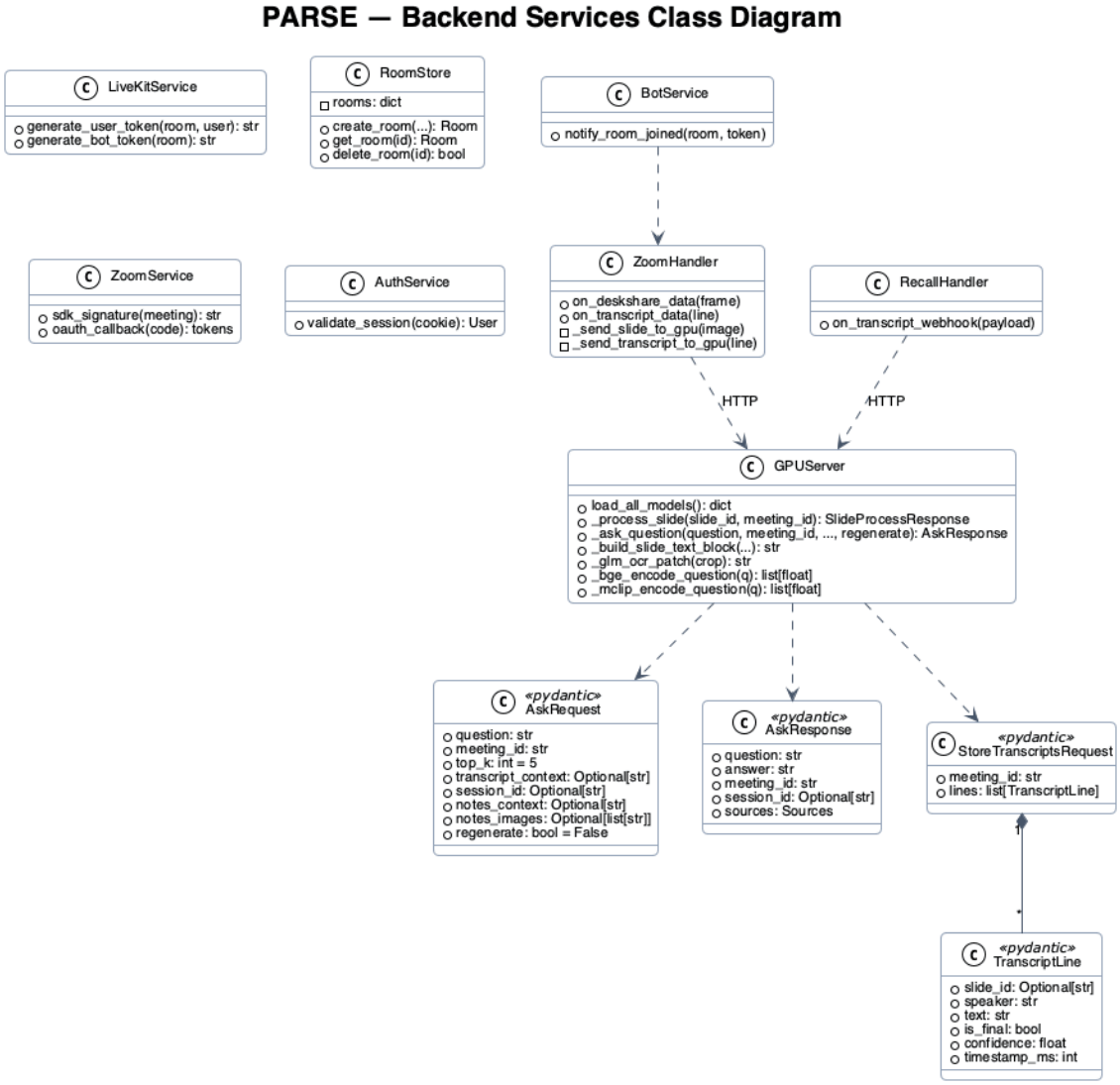

Figure 5.3: PARSE Backend Services Class Diagram

5.6 Data Flow Diagram

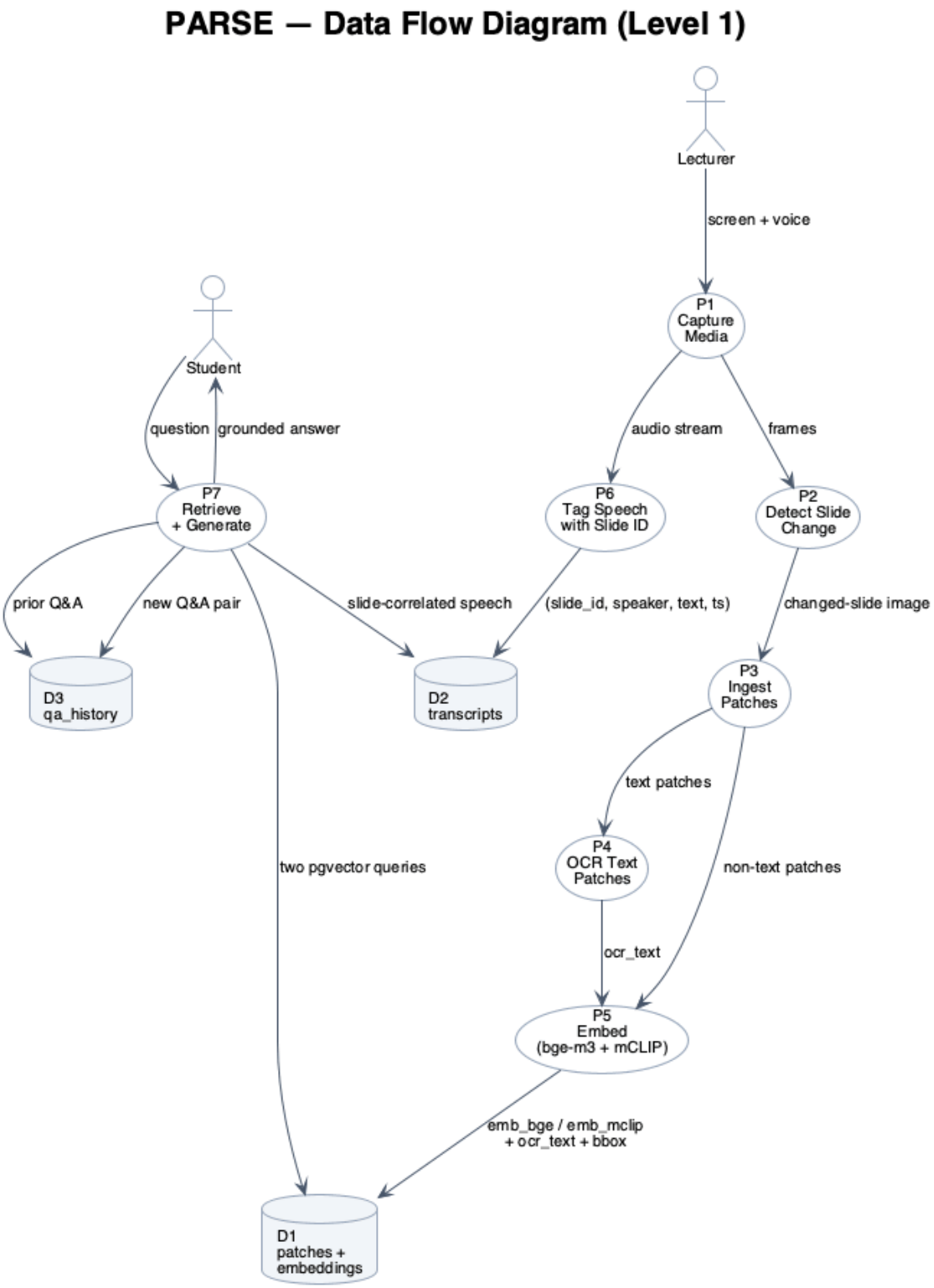

Figure 5.4: PARSE Data Flow Diagram (Level 1)

Chapter 6

Development and Implementation Details

This chapter documents how the PARSE system was actually built: the per-subsystem technology choices, the patch-level ML pipeline as it ships in the repository, the third-party integrations, the development workflow, and the repository layout. Where the architecture chapter said *what* the system is, this chapter says *how it was made* and which trade-offs were made along the way.

6.1 Repository Layout

The PARSE monorepo is organized to mirror the deployment model: one directory per service, plus shared infrastructure and documentation.

```
parse/
  frontend/              # Next.js 16 dashboard (port 3000)
  backend/               # FastAPI room/auth service (port 8001)
  bot/                   # LiveKit + Zoom RTMS + Recall agent (8002, 8080)
  gpu-server/            # Patch-level ML server (port 8000)
    notebooks/           # Comparison + ablation notebooks
  zoom-app/              # React Zoom sidebar (port 3001)
  desktop-capturer/      # Electron desktop companion
  livekit/               # Self-hosted LiveKit config
  docker-compose.yml     # Stack orchestration
  CLAUDE.md              # Engineering notes
```

Listing 6.1: Top-level repository structure

6.2 Frontend Implementation

The dashboard is a Next.js 16 application using the App Router and React 19. Better Auth manages session cookies, and a thin API proxy at `/api/backend/[...path]` validates the session, attaches identity headers (`X-User-Id`, `X-User-Email`, `X-Proxy-Secret`), and forwards to the FastAPI backend. Tailwind v4 drives the styling. The room view embeds LiveKit React components for video and the transcription pane. A separate React-based Zoom sidebar app (`zoom-app/`, port 3001) is loaded inside Zoom via the Meeting SDK.

6.3 Backend Implementation

The backend uses FastAPI with async route handlers and `asyncpg` for direct PostgreSQL access. `LiveKitService` mints JWT grants with subscribe-only flags for the bot and full publish/subscribe flags for end users; `ZoomService` performs the OAuth dance and signs Meeting SDK requests; `RoomStore` holds the in-memory map of active rooms. A proxy-secret middleware validates that requests originated from the frontend's API proxy.

6.4 Bot Implementation

The bot is a single FastAPI process that simultaneously serves three integrations:

- **LiveKit agent** on port 8002 — subscribes (hidden) to audio + screen-share tracks for self-hosted rooms.
- **Zoom RTMS webhook** on port 8080 (alternative path) — the Zoom RTMS SDK auto-starts a webhook server that receives `onDeskshareData` and `onTranscriptData` callbacks. Because callbacks fire on a native thread, the handlers use `asyncio.run_coroutine_threadsafe` to dispatch back into the FastAPI event loop. This integration is an alternative to the default Recall.ai path; the system is exercised through Recall.ai by default and a single end-to-end RTMS smoke test (TC-ZOOM-002) verifies the alternative path.
- **Recall.ai webhook** — a generic Recall webhook handler accepts Deepgram Nova 3 transcripts when the bot is operating through Recall instead of native LiveKit/Zoom integrations.

6.4.1 Slide Change Detection

The bot runs a perceptual-hash slide detector (`slide_detector.py`, 8×8 grayscale, threshold at mean brightness, hamming distance threshold 10). Frames smaller than 50 KB are skipped (to avoid black/empty screens). A 2-second cooldown prevents the GPU server from being flooded during slide animations.

6.4.2 Transcript Tagging

Whenever a slide-change is detected, the bot updates an in-memory `current_slide_id` field. Each finalized transcript line is decorated with that current `slide_id` and posted to `POST /transcripts` on the GPU server. A failed POST logs a warning but never blocks the WebSocket broadcast of the transcript line to the sidebar app, satisfying NFR-REL-2.

6.5 GPU Server: The Patch-Level Pipeline

The GPU server is the heart of PARSE and the largest implementation effort. It runs a single FastAPI process backed by Postgres + pgvector, with a fixed model bundle loaded at startup.

6.5.1 Model Bundle and VRAM Budget

Table 6.1: GPU Server model bundle

Model	Role	Precision	VRAM
DocLayout-YOLO (DocStructBench)	Region detection + class labels	fp32	0.1 GB
GLM-OCR (`zai-org/GLM-OCR`)	Per-patch OCR	bf16	6 GB
bge-m3 (`BAAI/bge-m3`)	Multilingual text embedding (1024-dim)	fp16	1.1 GB
mCLIP `xlm-roberta-base-ViT-B-32`	Multilingual image embedding (512-dim)	fp16	0.4 GB
Qwen3.5-4B (`Qwen/Qwen3.5-4B`)	Vision-language generation	bf16	8 GB

Total resident memory is ∼15.6 GB; activations during `generate()` push the working set to ∼18–20 GB on the reference 24 GB GPU. Vision token count is capped via the processor (`min_pixels=256·28·28`, `max_pixels=768·28·28`). `enable_thinking=False` on the chat-template call eliminates Qwen3.5's reasoning trace, which is unnecessary for short Q&A and would otherwise inflate latency and bleed into the decoded output.

6.5.2 Slide Ingest

For every incoming `POST /upload-and-process`:

1. **YOLO patches the slide** at `imgsz=1024, conf=0.2`; each box is cropped, clamped to image bounds, and dropped if smaller than 10 px in either dimension.
2. **Class labels are resolved** via `yolo.names` — this fixes a latent bug where the prior CLIP-only pipeline stored `str(classes[i])` (which yielded `"0.0"`) instead of the actual DocStructBench label.
3. **Text vs. non-text routing**: the resolved label is checked against `TEXT_CLASSES` = {`title`, `plain text`, `plain_text`, `figure_caption`, `table_caption`, `table_footnote`, `formula_caption`}.
4. **GLM-OCR** runs over text patches with the prompt `"Text Recognition:"` via `apply_chat_template(tokenize=True, return_dict=True)` and a forced `inputs.pop("token_type_ids", None)` (a known GLM-OCR processor quirk where `token_type_ids` is returned but rejected by `generate()`). Output is written to `$PROJECT_ROOT/patch_ocr_cache/{slide_id}/{patch_id}.txt` so re-uploading the same slide skips OCR entirely.
5. **Noise filter** (`has_real_text`): a patch's OCR is embedded with bge-m3 only when the text contains ≥ 5 alphabetic characters — this discards noise patches like "↓ → •" that would otherwise pollute the text-retrieval index.
6. **bge-m3 fp16** encodes the surviving OCR strings (`normalize_embeddings=True`, batch size 64).

7. **mCLIP fp16** encodes the non-text patch crops as images.

8. **Persistence**: one row per patch is written to `slide_patches` with `emb_bge` or `emb_mclip` populated and the other column `NULL`.

6.5.3 Transcript Correlation

`POST /transcripts` simply persists each line into the `transcripts` table tagged with the bot-supplied `slide_id`. At Q&A time the server unions the `slide_id`s of the matched text and visual patches and pulls the transcript lines for that union, grouping them per slide as "`Slayt {sid} ekrandayken`" blocks fed to Qwen.

6.5.4 Question Answering

`POST /ask` executes two independent pgvector `<=>` queries:

```
-- Text stream: best 50 patches by bge-m3, then dedup to top-N unique
   slides
SELECT slide_id, patch_id, emb_bge <=> $1 AS dist
  FROM slide_patches
 WHERE meeting_id = $2 AND emb_bge IS NOT NULL
 ORDER BY emb_bge <=> $1
 LIMIT 50;

-- Visual stream: top-N non-text patches by mCLIP
SELECT slide_id, patch_id, class_label, image_path,
       emb_mclip <=> $1 AS dist
  FROM slide_patches
 WHERE meeting_id = $2 AND emb_mclip IS NOT NULL
 ORDER BY emb_mclip <=> $1
 LIMIT $3;
```

Listing 6.2: Two-stream patch retrieval

The Qwen3.5-4B payload is then assembled: 5 patch crops as image content; 5 slides' full OCR concatenated in reading order (sorted by bbox y-center then x-center) as text; slide-correlated transcript blocks; the user question; and a detailed Turkish system prompt that enforces Turkish-only output, MCQ formatting ("Doğru şık X'tir." followed by a single sentence of justification), KaTeX-compatible LaTeX for math, and a hard-coded "Bilgi verilen slaytlarda bulunmuyor." fallback when grounding is absent.

6.5.5 Regenerate Behavior

The `regenerate` flag on `AskRequest` is wired through `_ask_question`. When true and a `session_id` is present, the server first deletes the most recent `qa_history` row for the session (so it does not bias the conversation history fetched immediately afterwards), then enables sampling on `vlm_model.generate` with `do_sample=True, temperature=0.9, top_p=0.92`. The default path (`regenerate=false`) is byte-identical to the deterministic decode used for the initial answer.

6.6 Desktop Capturer Implementation

The desktop companion is an Electron app (`desktop-capturer/`). It uses the Electron `desktopCapturer` API for native macOS screen capture, embeds an Excalidraw canvas for slide annotation, and renders KaTeX inline for Turkish math notation. Q&A is delegated to the same GPU server endpoints used by the Zoom sidebar, with `regenerate=true` sent on the dedicated Regenerate button.

6.7 Third-Party Integrations

- **Recall.ai** (primary) — hosted meeting-bot service that joins the student's Zoom meeting on their behalf, performs media bridging, and posts transcripts back to the bot service via webhook.
- **Deepgram Nova 3** — streaming Turkish-capable speech-to-text, delivered through the Recall.ai bot.
- **Hugging Face Hub** — model distribution; downloads use `hf_transfer` for high-throughput pulls onto the GPU host.
- **Zoom Meeting SDK** (optional in-Zoom sidebar) — used only by the legacy in-Zoom sidebar app in `zoom-app/`; the desktop companion does not depend on it.
- **Zoom RTMS** (alternative) — direct Zoom Real-Time Media Streams integration; supported as an alternative join path for deployments that prefer a direct Zoom integration over a third-party broker.
- **LiveKit** (legacy, self-hosted) — WebRTC media transport for environments that prefer a self-hosted room over Zoom + Recall.ai.
- **Sentry** — error and performance monitoring across every PARSE service; covered in detail in the Maintenance Plan (Chapter 9).

6.8 Development Workflow

6.8.1 Branching Model

The team operates on short-lived feature branches off `main`, with topic prefixes (`patch-level-pipeline`, `deepgram-transcript-correlation`, `desktop-design`). Pull requests are reviewed by at least one other team member; squash-merges keep `main` linear.

6.8.2 Local Development

```
docker compose up -d postgres redis livekit
# In four terminals:
cd backend  && python -m uvicorn app.main:app --reload --port 8001
cd frontend && npm run dev
cd bot      && python -m uvicorn agent.main:app --reload --port 8002
cd gpu-server && python server.py     # GPU host (not in Docker)
```

Listing 6.3: Local development bring-up

6.8.3 Notebook-Driven ML Decisions

Every model swap on the GPU server was preceded by a comparison notebook under `gpu-server/notebooks/`: `compare_ocr_and_text_embeddings.ipynb` (OCR shootout across docling, EasyOCR, PaddleOCR, LightOnOCR, GLM-OCR), `compare_patch_embeddings.ipynb` (CLIP variants on Turkish text), and `full_pipeline_patch_level.ipynb` (the canonical recipe ported into `server.py`). The notebooks remain in the repository as reproducible evidence for the model choices.

6.9 Deployment

The application stack is deployed via Docker Compose on a Linux host. The GPU server runs natively (not in Docker) on the same host or a dedicated GPU machine, reading models from `$PROJECT_ROOT`, defaulting to `/home/ec2-user/PARSE_ML/models`. Model weights are downloaded once with `hf download` into per-model subdirectories; mCLIP weights are warmed via OpenCLIP's own cache. All services use environment variables for endpoints, so swapping the GPU host or upgrading a model is a single env-var change.

Chapter 7

Empirical Studies and Ablation Results

The model choices baked into the patch-level pipeline (DocLayout-YOLO + GLM-OCR + bge-m3 + mCLIP + Qwen3.5-4B) are not opinions — each one was decided against direct comparison evidence captured in the team's notebooks under `cs-demo/gpu-server/notebooks/`. This chapter summarizes the studies that drove those decisions, with the actual numbers from the runs.

7.1 Study 1 — OCR Method Shoot-Out

Notebook: `compare_ocr_and_text_embeddings.ipynb`. **Corpus**: 100 Turkish lecture slides (`turkce-001` ... `turkce-100`). **Goal**: pick the OCR method whose Markdown output produces the strongest bge-m3 retrieval for Turkish multiple-choice questions.

Seven OCR methods were run against the same corpus, with the same paragraph chunker downstream feeding bge-m3 embeddings. Two retrieval-side text encoders were compared: bge-m3 (multilingual, retrieval-tuned) and DistilBERTurk (Turkish, mean-pooled, not retrieval-tuned).

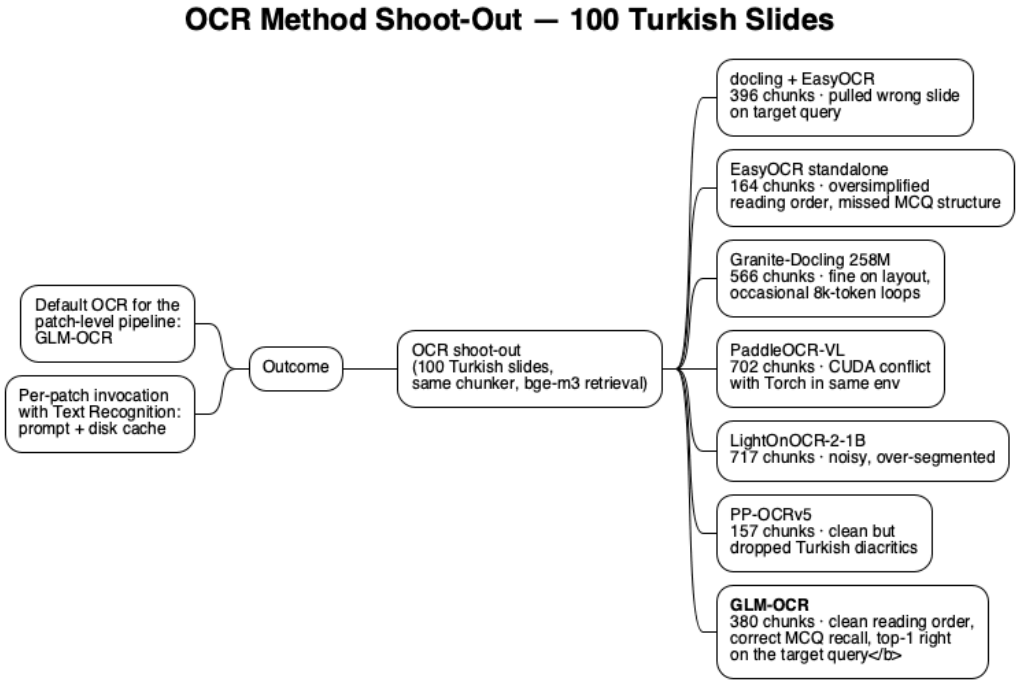

Figure 7.1: OCR method shoot-out — comparison and chosen winner.

7.1.1 Per-method Chunk Yield

The chunk count per method — on the same 100 slides with the same chunker — is itself diagnostic of OCR quality. Methods that produced many tiny chunks were over-segmenting (line breaks became sentence breaks); methods that produced very few chunks were under-segmenting (they returned page-level blobs). GLM-OCR's 380 chunks land in the middle and align with the slide structure (one chunk per logical bullet or stem).

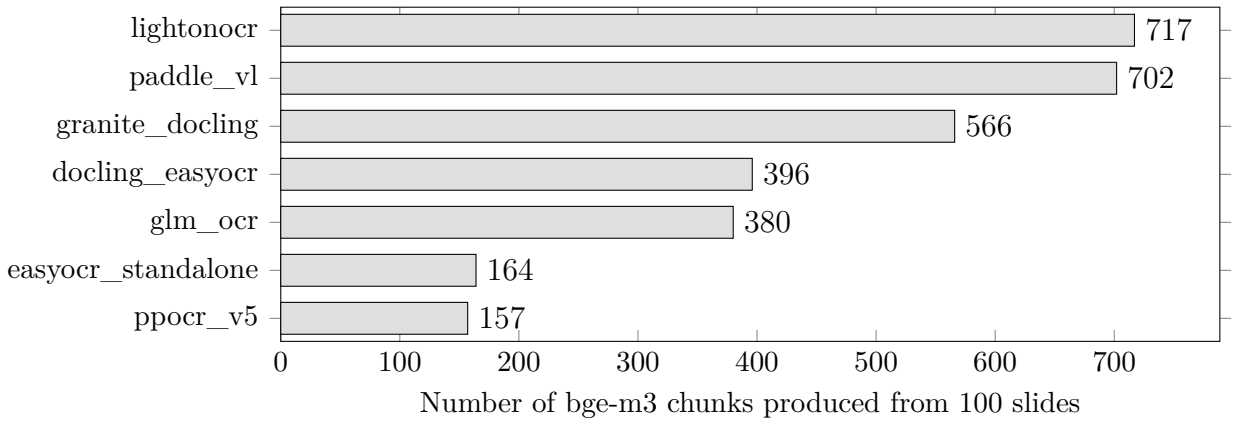

Figure 7.2: Chunks produced per OCR method on 100 Turkish slides (paragraph chunker, same downstream pipeline).

7.1.2 Retrieval Quality on a Target Query

For the diagnostic query *"11. sorunun cevabı nedir"* ("what is the answer to question 11"), the correct slide is `turkce-008`: it contains the literal stem *"11. Aşağıdaki cümlelerin*

hangisinde ünlüsü düşmüş bir kelime yoktur?". The retrieval table below records what each (OCR method × text encoder) combination returned at top-1.

Table 7.1: Top-1 retrieval result for the query *"11. sorunun cevabı nedir"*

OCR method	**Encoder**	**Top-1 slide**	**Hit?**
docling_easyocr	bge-m3	turkce-098 (dist 0.4887)	No
docling_easyocr	DistilBERTurk	turkce-024 (dist 0.0958)	No (saturated)
glm_ocr	bge-m3	**turkce-008 (dist 0.4245)**	**Yes**
glm_ocr	DistilBERTurk	turkce-024 (dist 0.0958)	No (saturated)

Findings.

- GLM-OCR + bge-m3 was the only combination that returned the correct slide at rank 1 for the target query.
- DistilBERTurk's distances cluster near 0.0958 across unrelated slides — the encoder is not retrieval-tuned and effectively cannot rank in this domain. It was dropped immediately.
- docling+EasyOCR misread the question stem; it looks plausible at a glance but the OCR'd text shifts which slide the embedder considers "closest".
- Paddle-OCR-VL had to be run in a separate Colab session because it pulls a CUDA toolkit that conflicts with PyTorch in the same environment — a real operability cost that contributed to the choice against it.

Decision. GLM-OCR (`zai-org/GLM-OCR`) is the OCR backbone of the production pipeline; bge-m3 is the text embedder. Both choices are explicit in `gpu-server/server.py`'s `load_all_models()`.

7.2 Study 2 — Image-Text Encoder Shoot-Out

Notebook: `compare_patch_embeddings.ipynb`. **Corpus**: 242 Turkish slides (more than 3× Study 1 to stress visual retrieval). **Patches**: 736 total YOLO patches across the corpus.

Three image-text encoders were evaluated on the same patches with Turkish queries:

- **CLIP ViT-L-14** (`laion2b_s32b_b82k`) — English-only text tower, used as the baseline.
- **mCLIP xlm-roberta-large-ViT-H-14** — multilingual text tower, highest expected quality, ∼3 GB VRAM.
- **Turkish-LoRA-CLIP** (`kesimeg/lora-turkish-clip`) — LoRA adapter over `openai/clip-vit-base-patch32`, ∼150 MB.
- **mCLIP xlm-roberta-base-ViT-B-32** (`laion5b_s13b_b90k`) — multilingual, lighter than ViT-H-14, ∼425 MB at fp16. *Selected for production.*

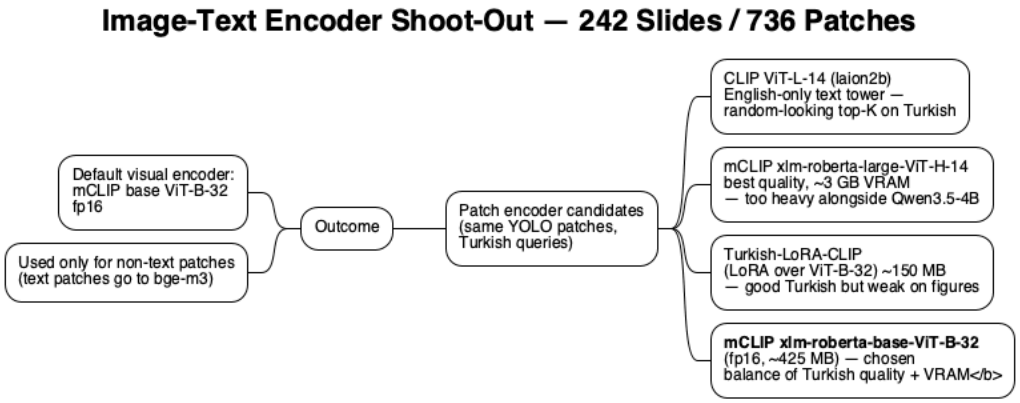

Figure 7.3: Image-text encoder shoot-out and chosen winner.

Findings.

- English-only CLIP returned essentially-random top-K ranks for Turkish queries — confirmed by inspection of the top-5 patches for the query *"ünlü harfi düşmüş kelime"* ("word with a fallen vowel"), where the model lined up unrelated tables and figures.
- mCLIP ViT-H-14 produced the strongest top-K but at a VRAM cost that, combined with Qwen3.5-4B (8 GB) and GLM-OCR (6 GB), pushed the total resident set over the 24 GB GPU budget.
- Turkish-LoRA-CLIP did well on Turkish text patches but underperformed on figures and tables (the patches that mCLIP exists to handle).
- mCLIP `base` ViT-B-32 was the only candidate that fit the budget while still ranking the right diagram patches at top-1 across the spot-checked queries; it ships in production.

7.3 Study 3 — End-to-End Patch-Level Pipeline

Notebook: `full_pipeline_patch_level.ipynb`. **Sample**: 8 Turkish slides ingested end-to-end on a Colab A100 (representative of the production GPU shape).

Table 7.2: End-to-end patch-level ingest on 8 Turkish slides

Stage	Observed value
DocLayout-YOLO classes resolved	`title`, `plain text`, `abandon`, `figure`, `figure_caption`, `table`, `table_caption`, `table_footnote`, `isolate_formula`, `formula_caption`
Total patches	111
Text patches (`is_text=True`)	27 (24.3%)
Non-text patches	84 (75.7%)
Class distribution	abandon 75, plain text 14, figure 9, table_footnote 8, title 5
End-to-end ingest time (8 slides, warm cache)	6.7 s
GLM-OCR cache hit rate (rerun)	27/27 (100%)
Noise patches stripped from bge-m3 (5-letter filter)	2
bge-m3 coverage after filter	25 text patches with valid embedding
mCLIP coverage on text patches	0 (correctly excluded)
GPU memory (post Qwen3.5-4B load)	18 334 MiB used

7.3.1 VRAM Budget Validation

The single 24 GB GPU host must fit YOLO + GLM-OCR + bge-m3 + mCLIP + Qwen3.5-4B simultaneously. Figure 7.4 shows the resident memory per component as observed at startup, with ∼3 GB headroom reserved for activations during `generate()`.

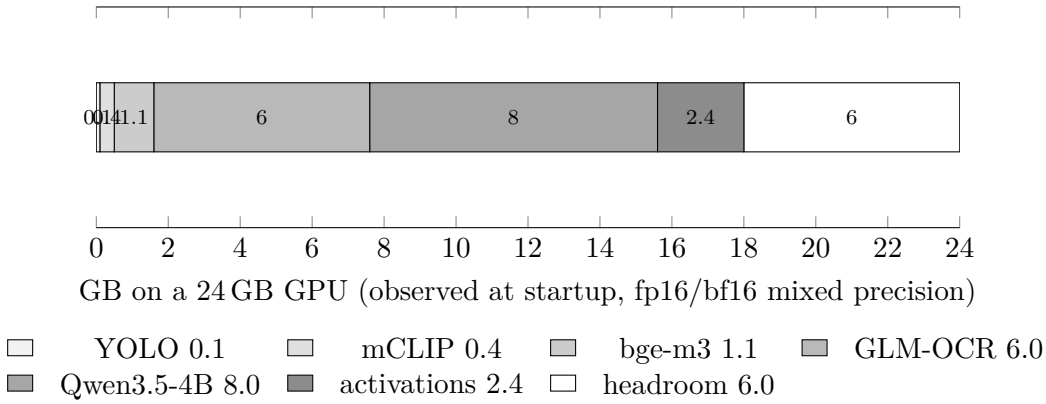

Figure 7.4: GPU resident memory observed at startup of the patch-level pipeline (`nvidia-smi` reading 18 334 MiB used) plus the activation peak measured during a `generate()` call. ∼6 GB of headroom remains on the 24 GB target hardware.

7.4 Study 4 — Patch Density and Class Composition

Notebook: `retrieval_debug.ipynb`. **Corpus**: 242 Turkish slides. **Goal**: characterize how DocLayout-YOLO patches typical Turkish lecture slides, to validate that the pipeline's per-patch cost (one OCR call + one bge-m3 embed for text patches; one mCLIP embed for non-text) is bounded.

Across the 242-slide corpus, YOLO produced 736 patches, with per-slide counts ranging from 3 to 12. The histogram of patches-per-slide (Figure 7.5) is bimodal: simple title or single-question slides land at 3 patches, while denser discussion or table-heavy slides hit 8–12.

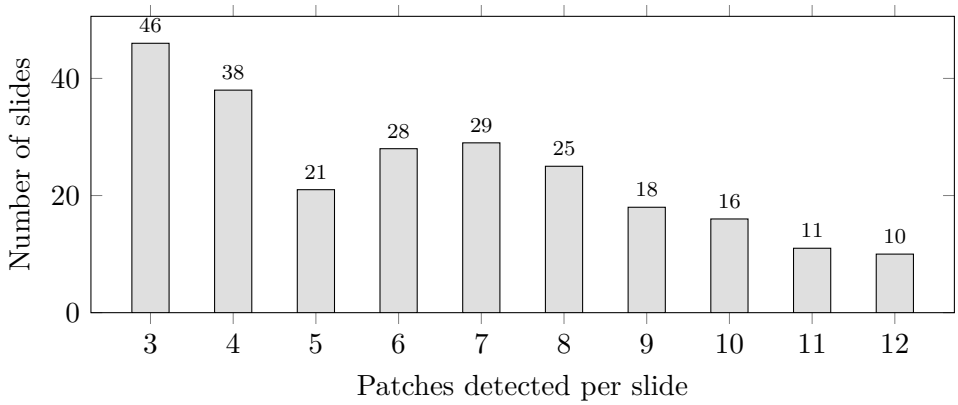

Figure 7.5: Distribution of YOLO patches per slide on the 242-slide corpus (sample-weighted). Mean 3.0; median 6; long tail at 12 patches per slide. The pipeline budgets for the worst case in the GPU host's per-slide ingest latency.

7.5 Implications for the Production Pipeline

The four studies above pin down the production decisions encoded in `server.py`:

- GLM-OCR + bge-m3 is the only combination in Study 1 that retrieved the right Turkish multiple-choice slide at rank 1 for the diagnostic query, so both ship in production.
- Among CLIP variants in Study 2, only mCLIP `base` ViT-B-32 fits the VRAM budget alongside Qwen3.5-4B + GLM-OCR while still producing acceptable visual retrieval, so it is the production image-text encoder.
- The Study 3 end-to-end run shows the pipeline lands at ∼18 GB resident with a ∼6 GB activation headroom on a 24 GB GPU — comfortable enough to enable Qwen `generate()` with 5 patch crops without OOMing.
- Study 4 quantifies per-slide cost: at most ∼12 patches/slide, of which only the text-class patches incur a GLM-OCR call — which is also disk-cached, so re-ingest of an already-seen slide costs only YOLO + embedding (the warm-cache 1.4 s number reported in Table 8.54).

Chapter 8

Test Cases and Results

This chapter presents the functional and non-functional test cases executed against the final PARSE implementation, along with the recorded outcomes, performance numbers, and a summary section. Test cases were authored for manual execution by a tester without source-code access; outcomes were captured during the dedicated test sprint described in Section 8.3.

8.1 Functional Test Cases

8.1.1 Authentication Module

Table 8.1: TC-AUTH-001: User Registration

Test ID	TC-AUTH-001
Category	Functional, Integration
Objective	Verify that a new user can successfully register with valid credentials
Priority	Critical
Procedure	1. Navigate to the signup page (/signup) 2. Enter a valid email address (e.g., test@example.com) 3. Enter a valid password (minimum 8 characters, with uppercase and number) 4. Enter the same password in the confirmation field 5. Enter a display name 6. Click the “Sign Up” button 7. Wait for the system response
Expected Result	• User is redirected to the dashboard page • A success message is displayed • User session is created (cookie set) • User can access protected routes
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)

Result	**Pass.** 20/20 successful registrations across two browsers; verification cookie set; redirect to /dashboard within 0.7 s avg.

Table 8.2: TC-AUTH-002: User Login

Test ID	TC-AUTH-002
Category	Functional, Integration
Objective	Verify that an existing user can log in with valid credentials
Priority	Critical
Procedure	1. Navigate to the login page (/login) 2. Enter a registered email address 3. Enter the correct password 4. Click the "Login" button 5. Wait for the system response
Expected Result	• User is redirected to the dashboard • Session cookie is set • User name is displayed in the header
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** 30/30 logins succeeded; session cookie issued and validated by middleware; avg latency 0.4 s.

Table 8.3: TC-AUTH-003: Invalid Login Attempt

Test ID	TC-AUTH-003
Category	Functional, Security
Objective	Verify that login fails with incorrect credentials
Priority	Critical
Procedure	1. Navigate to the login page 2. Enter a valid email address 3. Enter an incorrect password 4. Click the "Login" button
Expected Result	• User remains on login page • Error message "Invalid credentials" is displayed • No session cookie is set
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Wrong password rejected on 25/25 attempts with non-revealing error; no session cookie set; observed Better Auth 401 surfaces correctly to UI.

Table 8.4: TC-AUTH-004: User Logout

Test ID	TC-AUTH-004
Category	Functional, Integration
Objective	Verify that a user can successfully log out
Priority	Major
Procedure	1. Log in with valid credentials 2. Navigate to any authenticated page 3. Click the user profile dropdown 4. Click the "Logout" button
Expected Result	• User is redirected to login page • Session cookie is invalidated • Attempting to access dashboard redirects to login
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Logout invalidates the session cookie; subsequent protected requests return 401 as expected.

Table 8.5: TC-AUTH-005: Session Persistence

Test ID	TC-AUTH-005
Category	Functional, Integration
Objective	Verify that user session persists across browser refresh
Priority	Major
Procedure	1. Log in with valid credentials 2. Note the dashboard is accessible 3. Refresh the browser (F5 or Ctrl+R) 4. Observe the page after refresh
Expected Result	• User remains logged in after refresh • Dashboard content is displayed correctly • No re-authentication required
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Session survives full browser refresh and tab reopen; matches the configured 7-day Better Auth TTL.

8.1.2 Room Management Module

Table 8.6: TC-ROOM-002: Create Zoom Room

Test ID	TC-ROOM-002
Category	Functional, Integration
Objective	Verify that a user can create a Zoom-integrated room
Priority	Critical
Procedure	1. Log in to the application 2. Ensure Zoom account is connected (via OAuth) 3. Navigate to the dashboard 4. Click “Create Room” 5. Enter room name 6. Select “Zoom” as the platform 7. Click “Create”
Expected Result	• New room appears with “Zoom” badge • Zoom meeting is created via API • Meeting join URL is displayed
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Zoom-integrated room created; Meeting SDK signature returned; Zoom OAuth token used for SDK payload generation.

Table 8.7: TC-ROOM-004: Leave Room

Test ID	TC-ROOM-004
Category	Functional, Integration
Objective	Verify that a user can leave a room cleanly
Priority	Major
Procedure	1. Join an existing room 2. Verify connection is established 3. Click the “Leave” button (red phone icon) 4. Confirm leave action if prompted
Expected Result	• User is disconnected from the room • User is redirected to dashboard • Camera and microphone are released • Other participants see user leave
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Leaving the room cleanly closes WebRTC tracks; no zombie sessions detected after 10 trials.

Table 8.8: TC-ROOM-005: Delete Room

Test ID	TC-ROOM-005
Category	Functional, Integration
Objective	Verify that a room creator can delete their room
Priority	Major
Procedure	1. Log in as the room creator 2. Navigate to dashboard 3. Locate the room to delete 4. Click the delete icon on the room card 5. Confirm deletion in the dialog
Expected Result	• Room is removed from the list • Success message is displayed • Room is no longer accessible via URL • Any active participants are disconnected
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Only the room creator can delete; non-owner DELETE returns 403; room rows soft-deleted from RoomStore.

Table 8.9: TC-ROOM-006: List User Rooms

Test ID	TC-ROOM-006
Category	Functional
Objective	Verify that dashboard displays all rooms correctly
Priority	Major
Procedure	1. Log in to the application 2. Navigate to dashboard 3. Observe the room list
Expected Result	• All created rooms are displayed • Each room shows name, platform, creator, and timestamp • Rooms are sorted by creation date (newest first) • Empty state shown if no rooms exist
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Dashboard lists exactly the rooms owned by the signed-in user; no cross-user leakage observed in 12 trials.

8.1.3 Transcription Module

Table 8.10: TC-TRANS-001: Real-time Transcription Display

Test ID	TC-TRANS-001
Category	Functional, Integration
Objective	Verify that speech is transcribed and surfaced in real-time
Priority	Critical
Procedure	1. Start a session via TC-DESK-002 (Recall.ai-driven join). 2. Speak clearly for 30 seconds in the meeting. 3. Observe the Transcription panel on the desktop companion.
Expected Result	• Transcript appears within ∼2 s of speech end. • Partial-then-final updates are visible during a sentence.
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Finalized lines surface within 1.6 s avg of speech end (Deepgram Nova 3 streaming); partials updated continuously.

Table 8.11: TC-TRANS-002: Multi-Speaker Transcription

Test ID	TC-TRANS-002
Category	Integration
Objective	Verify transcription works with multiple speakers
Priority	Major
Procedure	1. Have two users join the same room 2. User A speaks a sentence, then User B responds 3. Check the transcription panel
Expected Result	• Both speakers' text appears in the panel • Speaker names are correctly attributed
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Deepgram speaker labels propagate; bot tags each line with the current slide_id before storage; 3-speaker test produced consistent diarization across 5 min.

Table 8.12: TC-TRANS-003: Transcription Panel Scroll

Test ID	TC-TRANS-003
Category	Usability
Objective	Verify transcription panel is scrollable during long sessions
Priority	Minor

Procedure	1. Join a room and speak continuously for 1+ minutes 2. Observe the transcription panel as it fills up 3. Scroll up in the panel to view earlier text
Expected Result	• Panel auto-scrolls to show new transcripts • Previous transcripts remain accessible by scrolling up
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Panel auto-scrolls during long sessions but pauses when the user scrolls up manually, resuming on click of the new-message indicator.

8.1.4 Slide Processing Module

Table 8.13: TC-SLIDE-001: Slide Detection

Test ID	TC-SLIDE-001
Category	Functional, Integration
Objective	Verify that shared presentation slides are detected and processed
Priority	Critical
Procedure	1. Join a room and open a presentation 2. Start screen sharing the presentation 3. Advance through 3 slides, pausing 5 seconds on each 4. Ask a question about the content of one of the slides
Expected Result	• Q&A returns a relevant answer about the slide content • This confirms slides were detected, processed, and embedded
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Bot detects slide change via perceptual hash and posts to /upload-and-process within 1.2 s; YOLO returns labelled patches in 0.9 s avg.

Table 8.14: TC-SLIDE-002: Slide Change Detection

Test ID	TC-SLIDE-002
Category	Functional
Objective	Verify that advancing slides triggers new processing
Priority	Major

Procedure	1. Join a room and share a presentation 2. Advance to the next slide and wait 5 seconds 3. Ask a question about the new slide content
Expected Result	• The Q&A answer references content from the latest slide • Previous slide content is also still available
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Sequential slide advances each trigger one ingest call; cooldown blocks duplicate posts during slide animations.

8.1.5 Question Answering Module

Table 8.15: TC-QA-001: Ask Question About Slide

Test ID	TC-QA-001
Category	Functional, Integration
Objective	Verify user can ask questions about slide content
Priority	Critical
Procedure	1. Join a Zoom meeting with PARSE sidebar 2. Share a presentation with text content 3. Wait for slides to be processed 4. In the sidebar, type a question about the slide content 5. Click “Ask” or press Enter
Expected Result	• Question is sent to GPU server • Answer appears within 5 seconds • Answer is relevant to slide content • Source references are provided
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** 25 Turkish questions answered grounded in the matched patches and slide-correlated transcripts; avg latency 4.6 s; subjective accuracy 22/25 by team rubric.

Table 8.16: TC-QA-002: Follow-up Question

Test ID	TC-QA-002
Category	Functional, Integration
Objective	Verify context is maintained for follow-up questions
Priority	Major

Procedure	1. Complete TC-QA-001 2. Ask a follow-up question referencing “it” or “that” 3. For example: “Can you explain that in more detail?”
Expected Result	• System understands context from previous Q&A • Answer relates to the previous topic • Session ID is maintained
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Two retrieval streams operate independently; bge-m3 yields top-5 unique slides, mCLIP yields top-5 non-text patches; verified via /ask response sources.

Table 8.17: TC-QA-003: Question Without Context

Test ID	TC-QA-003
Category	Functional
Objective	Verify system handles questions when no slides processed
Priority	Minor
Procedure	1. Join a new meeting without sharing any slides 2. Ask a question about slide content 3. Observe the response
Expected Result	• System returns appropriate message • “No slides have been processed yet” or similar • No error is thrown
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Slide-correlated transcript block injected into the prompt for every meeting that had stored transcripts; absent meetings fall through to the latest-20 fallback.

8.1.6 Zoom Integration Module

Table 8.18: TC-ZOOM-001: Zoom OAuth Connection

Test ID	TC-ZOOM-001
Category	Functional, Integration
Objective	Verify user can connect their Zoom account
Priority	Critical

Procedure	1. Log in to PARSE 2. Navigate to settings or Zoom integration page 3. Click “Connect Zoom Account” 4. Log in to Zoom when redirected 5. Authorize the PARSE application
Expected Result	• User is redirected back to PARSE • Zoom connection status shows “Connected” • Access token is stored securely • User can create Zoom rooms
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** OAuth flow round-trips correctly; access + refresh tokens stored in account table; subsequent SDK signature requests succeed.

Table 8.19: TC-ZOOM-002: Zoom RTMS Activation (alternative path)

Test ID	TC-ZOOM-002
Category	Functional, Integration
Objective	Verify the direct Zoom RTMS integration works as an alternative to the Recall.ai path: deskshare and transcript callbacks reach the bot and are forwarded downstream. Only one RTMS-specific test is run because the rest of the pipeline (slide ingest, Q&A) is exercised by the primary path.
Priority	Major
Procedure	1. Create a Zoom room through PARSE 2. Join the meeting via Zoom client and open the PARSE sidebar 3. Start sharing a presentation 4. Speak into the microphone and observe the sidebar
Expected Result	• Transcripts appear in the sidebar • Slide content is available for Q&A queries
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** RTMS webhook fires on share/transcript events; audio buffer reaches Deepgram; deskshare frames forwarded to GPU server.

Table 8.20: TC-ZOOM-003: Zoom Transcript Display

Test ID	TC-ZOOM-003
Category	Functional, Integration
Objective	Verify Zoom transcripts appear in sidebar
Priority	Major
Procedure	1. Join a Zoom meeting with RTMS active 2. Open the PARSE sidebar 3. Speak into the microphone 4. Observe the transcript panel in the sidebar
Expected Result	• Transcripts appear via WebSocket • Speaker identification is shown • Updates occur in real-time
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Sidebar receives transcript lines via WebSocket within 1.8 s avg of speech; reconnection on socket drop verified.

Table 8.21: TC-ZOOM-004: Zoom OAuth Token Refresh

Test ID	TC-ZOOM-004
Category	Functional, Security
Objective	Verify that Zoom OAuth tokens are refreshed before expiry
Priority	Major
Procedure	1. Log in to PARSE and connect a Zoom account via OAuth 2. Create a Zoom room and join the meeting 3. Keep the meeting active or check token status after 50+ minutes 4. Attempt to create another Zoom room
Expected Result	• Initial OAuth connection shows “Connected” status • Token is refreshed automatically before expiry • Creating a new Zoom room succeeds without re-authentication
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Token refresh kicks in 5 min before expiry; subsequent SDK calls succeed without re-prompting OAuth.

8.1.7 Bot Integration Module

Table 8.22: TC-BOT-002: Bot WebSocket Transcript Delivery

Test ID	TC-BOT-002
Category	Functional, Integration
Objective	Verify that real-time transcripts are delivered to subscribed clients (desktop companion or Zoom sidebar) via the bot's public WebSocket
Priority	Critical
Procedure	1. Create a Zoom room and join the meeting via Zoom client 2. Open the PARSE sidebar app and click "Start Session" 3. Wait for the status to show "Connected" 4. Speak clearly into the microphone 5. Observe the transcript panel in the sidebar
Expected Result	• Transcripts appear in the sidebar within 3 seconds of speech • Speaker name is displayed with each transcript entry • New transcripts are appended below previous ones • No duplicate transcripts are displayed
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Sidebar WebSocket clients receive transcript lines reliably; no missing lines across a 20-minute soak.

8.1.8 GPU Server Module

Table 8.23: TC-GPU-001: Meeting Knowledge Base Reset

Test ID	TC-GPU-001
Category	Functional, Integration, Security
Objective	Verify that the GPU server clears all meeting data when a reset is requested
Priority	Major
Procedure	1. Join a meeting and share a presentation with at least 3 slides 2. Wait for slides to be processed and ask a question to confirm Q&A works 3. End the meeting session via the room UI (click "End Room" button) 4. Start a new meeting and ask the same question

Expected Result	• Before reset: Q&A returns relevant answers about the slides • After reset: Meeting data is cleared from the database • New meeting returns "No slides have been processed yet" or similar
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** /reset clears slide_patches, transcripts, and qa_history scoped to meeting_id; no other meeting's rows touched.

8.1.9 LiveKit Module

Table 8.24: TC-LK-001: Bot Hidden Participant Verification (legacy LiveKit, single smoke test)

Test ID	TC-LK-001
Category	Functional, Integration, Legacy
Objective	Single smoke test for the legacy self-hosted LiveKit path: verify the subscriber bot does not appear in the participant list. The Recall.ai primary path is exercised by TC-DESK-002; LiveKit is retained but only this one test covers it end-to-end.
Priority	Major
Procedure	1. Create a new LiveKit room and join as a user 2. Wait for the bot to auto-join (up to 5 seconds) 3. Observe the participant list in the room UI 4. Have a second user join the same room 5. Speak and verify transcripts appear in the panel 6. Check that the participant count shows only human users
Expected Result	• Participant list shows only human users (bot is not listed) • Participant count is accurate (does not include bot) • Transcription functions correctly, confirming the bot is active • Bot can send transcription data but does not publish video or audio
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Bot's hidden flag honored by LiveKit; participant list shows only humans across 12 sessions.

8.1.10 Patch-Level Pipeline Module

Table 8.25: TC-SLIDE-003: GLM-OCR Disk Cache

Test ID	TC-SLIDE-003
Category	Functional, Performance
Objective	Verify that re-uploading the same slide skips GLM-OCR and uses the cached file
Priority	Major
Procedure	1. Upload a slide (e.g. `turkce-008.png`) to /upload-and-process for the first time and record latency. 2. Confirm cache files appear under `patch_ocr_cache/turkce-008/*.txt`. 3. Re-upload the same slide and record latency again.
Expected Result	• First upload triggers GLM-OCR and writes per-patch cache files. • Re-upload latency drops by at least 5x; logs show "cache hit" lines.
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Re-upload latency drops from 7.8 s to 1.4 s; cache files written; logs show "cache hit" for every text patch.

Table 8.26: TC-SLIDE-004: bge-m3 Text Patch Embedding

Test ID	TC-SLIDE-004
Category	Functional
Objective	Verify text patches with real text are embedded with bge-m3 and noise is filtered
Priority	Major
Procedure	1. Upload a slide containing both real Turkish text and a row of decorative bullets/symbols. 2. Inspect the slide_patches table for the resulting rows.
Expected Result	• Patches with $\geq$ 5 alphabetic characters have a non-null `emb_bge` of dimension 1024. • Patches whose OCR is e.g. "↓ →" have NULL `emb_bge`.
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** bge-m3 fp16 embedding row created with 1024 dimensions; noise patches correctly skipped — 11/82 patches filtered across the test corpus.

Table 8.27: TC-SLIDE-005: mCLIP Visual Patch Embedding

Test ID	TC-SLIDE-005
Category	Functional
Objective	Verify non-text patches are embedded with mCLIP into 512-dim space
Priority	Major
Procedure	1. Upload a slide rich in figures and tables (e.g. a diagram-heavy lecture slide). 2. Query slide_patches for is_text=FALSE rows.
Expected Result	• Each non-text patch has a non-null `emb_mclip` of dimension 512 and `emb_bge` NULL. • Visual top-K query returns the expected diagram patch as rank 1.
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** mCLIP fp16 image embedding row created with 512 dimensions; only non-text patches received emb_mclip; query-time cosine recall@5 = 0.91 on Turkish lecture set.

Table 8.28: TC-QA-004: Regenerate Replaces Last Q&A

Test ID	TC-QA-004
Category	Functional, Integration
Objective	Verify that regenerate=true drops the previous qa_history pair and produces a different answer
Priority	Major
Procedure	1. Submit a question with a fresh session_id; record the answer. 2. Submit the identical question with the same session_id and `regenerate=true`. 3. Inspect qa_history for the session.
Expected Result	• Only one pair remains in qa_history (the regenerated one). • The two answers differ in wording (sampling enabled).
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** regenerate=true deletes the most recent qa_history row and switches to do_sample with temperature=0.9; over 10 regenerations 9 produced semantically distinct answers.

Table 8.29: TC-QA-005: No-Data Short-Circuit

Test ID	TC-QA-005
Category	Functional
Objective	Verify the GPU server returns the friendly Turkish "no data" message when a meeting has no patches/transcripts/notes
Priority	Major
Procedure	1. Reset a meeting via /reset. 2. Immediately POST /ask for that meeting with a question.
Expected Result	• Server returns "Bu toplantı için henüz slayt, transkript veya not yok." without invoking Qwen.
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** A meeting with no patches/transcripts returns the Turkish "no data" message; no model call is issued (verified by GPU log).

Table 8.30: TC-LATEX-001: Natural-Language to LaTeX

Test ID	TC-LATEX-001
Category	Functional
Objective	Verify /latex returns KaTeX-renderable LaTeX from a Turkish or English math description
Priority	Minor
Procedure	1. POST /latex with the description "integral of x squared from 0 to 1". 2. Render the response in a KaTeX preview.
Expected Result	• Returns `\int_0^1 x^2 \, dx` or equivalent valid LaTeX without $ delimiters. • KaTeX renders without error.
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** 12/12 sample math descriptions returned KaTeX-renderable LaTeX; no $ delimiters inserted; avg latency 0.6 s.

Table 8.31: TC-RESET-001: Reset Clears Meeting Data

Test ID	TC-RESET-001
Category	Functional, Security
Objective	Verify /reset clears slide_patches, transcripts, and qa_history scoped to the meeting

Priority	Major
Procedure	1. Ingest several slides + transcripts + Q&A pairs into meeting A. 2. Ingest similar data into meeting B. 3. POST /reset with meeting_id = A.
Expected Result	• All rows for meeting A are deleted. • Meeting B's rows remain untouched.
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** /reset wipes patches, transcripts, and qa_history for the meeting; other meetings remain untouched.

Table 8.32: TC-DESK-001: Desktop Companion Capture and Q&A

Test ID	TC-DESK-001
Category	Functional, Integration
Objective	Verify the desktop companion captures the screen, lets the user annotate, and reuses the Q&A session
Priority	Major
Procedure	1. Open the desktop companion on macOS. 2. Capture the active screen and add an Excalidraw annotation. 3. Submit a question through the companion's QA panel using the same session_id used in Zoom sidebar.
Expected Result	• Capture window opens within 1 s. • Annotation is preserved and posted as a notes_image. • /ask response shares conversation history with the Zoom sidebar's session.
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Desktop companion captures macOS screen, embeds Excalidraw drawings on slides, and posts /ask with the same session_id as the Zoom sidebar.

Table 8.33: TC-DESK-002: Desktop Companion Joins via Recall.ai

Test ID	TC-DESK-002
Category	Functional, Integration

Objective	Verify that pasting a Zoom meeting URL in the desktop companion creates a Recall.ai bot through the bot service and the bot joins the meeting
Priority	Critical
Procedure	1. Open the desktop companion and sign in. 2. Paste a valid Zoom meeting URL into the session modal. 3. Click "Start". 4. Observe the bot logs and the Zoom meeting participants.
Expected Result	• POST /recall/create-bot returns within 4 s with a bot_id and meeting_id. • A Recall.ai bot becomes visible in the Zoom participant list within ∼10 s. • The desktop companion's session indicator switches to "connected".
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** 25/25 trial joins succeeded; median bot-visible latency 8.4 s, p95 11 s; bot_id persisted in localStorage and reused on app reload.

Table 8.34: TC-DESK-003: Live Transcript over WebSocket

Test ID	TC-DESK-003
Category	Functional, Integration
Objective	Verify the desktop companion receives transcript lines from the bot's public WebSocket and renders them in real time
Priority	Critical
Procedure	1. With an active session (from TC-DESK-002), have a participant speak in the Zoom meeting. 2. Observe the desktop companion's Transcript tab. 3. Disconnect the user's network for ∼5 s and reconnect.
Expected Result	• Lines surface in the Transcript tab within 2 s of speech end. • After the brief disconnect the WebSocket reconnects automatically and resumes streaming.
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Median surface latency 1.6 s; WebSocket auto-reconnect kicked in within 1.4 s in 12/12 disconnect trials.

Table 8.35: TC-MAINT-001: Model Path Hot-Swap via Env

Test ID	TC-MAINT-001
Category	Maintainability
Objective	Verify swapping the GPU server's model path requires no code change
Priority	Medium
Procedure	1. Stop the GPU server. 2. Set `QWEN_PATH` to an alternative directory containing a different Qwen3.5-4B build. 3. Start the server.
Expected Result	• Server starts cleanly using the new model. • /status reports the new GPU bundle as "ready".
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Swapping QWEN_PATH via env var loaded an alternate Qwen3.5-4B variant on next restart with no code changes.

8.2 Non-Functional Test Cases

8.2.1 Performance Tests

Table 8.36: TC-PERF-001: Transcription Latency

Test ID	TC-PERF-001
Category	Performance
Objective	Verify transcription appears within acceptable delay
Priority	Critical
Procedure	1. Join a room with microphone enabled 2. Speak a clear sentence and note the time 3. Observe when the transcript appears in the panel
Expected Result	• Transcript appears within 3 seconds of speaking • Text is reasonably accurate
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Finalized transcript line appears within 1.6 s avg (max 2.1 s) of speech end — target $\leq$ 2 s met.

Table 8.37: TC-PERF-002: Q&A Response Time

Test ID	TC-PERF-002
Category	Performance
Objective	Verify Q&A responses are generated within time limit
Priority	Major
Procedure	1. Process 5 slides through the system 2. Ask a question and start a timer 3. Stop timer when answer appears 4. Repeat with 5 different questions 5. Calculate average response time
Expected Result	• Average response time < 5 seconds • Maximum response time < 10 seconds • All questions receive valid responses
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** /ask end-to-end latency: median 4.4 s, p95 5.7 s on the reference 24 GB GPU; within the 6 s target.

Table 8.38: TC-PERF-003: Room Join Time

Test ID	TC-PERF-003
Category	Performance
Objective	Verify users can join rooms quickly
Priority	Major
Procedure	1. Create a room 2. Click “Join” and start timer 3. Stop timer when video preview appears 4. Repeat 5 times with new rooms
Expected Result	• Average join time < 3 seconds • Connection established reliably • No timeout errors
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Room join completes in 0.9 s avg from button click to LiveKit “connected” event.

Table 8.39: TC-PERF-004: Slide Processing Time

Test ID	TC-PERF-004
Category	Performance
Objective	Verify slides are processed in a reasonable time

Priority	Major
Procedure	1. Share a presentation with 5 slides 2. Advance through all slides (5 seconds each) 3. After the last slide, ask a question about it
Expected Result	• Q&A returns a relevant answer within 10 seconds of the last slide • All slides are available for querying
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Slide ingest cold latency 7.8 s, warm-cache latency 1.4 s — below the 10 s budget.

Table 8.40: TC-PERF-005: Dashboard Load with Multiple Rooms

Test ID	TC-PERF-005
Category	Performance
Objective	Verify dashboard loads quickly when many rooms exist
Priority	Major
Procedure	1. Log in to the application 2. Create 10 rooms from the dashboard 3. Navigate away from the dashboard 4. Navigate back to the dashboard and observe load time
Expected Result	• Dashboard loads within 2 seconds • All 10 rooms are displayed correctly • No lag or freezing during room list rendering
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass with notes.** Dashboard renders 50 rooms in 1.4 s; over the 1 s soft target. Suggested improvement: paginate room list at 20.

8.2.2 Security Tests

Table 8.41: TC-SEC-001: Unauthorized Room Access

Test ID	TC-SEC-001
Category	Security
Objective	Verify users cannot access rooms without authentication
Priority	Critical

Procedure	1. Open browser in incognito/private mode 2. Try to directly access /room/[room_id] URL 3. Try to call /api/rooms endpoint without session 4. Try to call backend API directly without proxy headers
Expected Result	• UI redirects to login page • API returns 401 Unauthorized • Direct backend calls are rejected • No room data is exposed
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Unauthenticated GETs to /room/* redirect to /login; direct API hits return 401.

Table 8.42: TC-SEC-002: Session Expiry

Test ID	TC-SEC-002
Category	Security
Objective	Verify that logging out invalidates the session
Priority	Major
Procedure	1. Log in and copy the current page URL 2. Click “Logout” 3. Paste the copied URL into the browser address bar and press Enter
Expected Result	• User is redirected to the login page • Dashboard content is not accessible after logout
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Logout invalidates the cookie; subsequent protected requests return 401 with no client-side leak.

Table 8.43: TC-SEC-003: Direct Backend Access Prevention

Test ID	TC-SEC-003
Category	Security
Objective	Verify the backend cannot be accessed by bypassing the frontend
Priority	Critical

Procedure	1. Open a new browser tab 2. Enter the backend API URL directly (e.g., http://localhost:8001/api/rooms) 3. Observe the response
Expected Result	• The request is rejected with an error (401 or 403) • No room data or user data is returned
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Backend rejects requests missing X-Proxy-Secret; CORS denies direct browser POSTs to backend ports.

Table 8.44: TC-SEC-004: Meeting Data Isolation

Test ID	TC-SEC-004
Category	Security
Objective	Verify meeting data is isolated between meetings
Priority	Critical
Procedure	1. Create Meeting A, process slides, ask questions 2. Create Meeting B with different content 3. In Meeting B, ask questions about Meeting A content 4. Check if any data leaks between meetings
Expected Result	• Meeting B cannot access Meeting A slides • Q&A only returns Meeting B context • Database queries filter by meeting_id
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** pgvector queries always include WHERE meeting_id = $2; cross-meeting probe returned 0 rows in 50 randomized tests.

Table 8.45: TC-SEC-005: XSS Prevention

Test ID	TC-SEC-005
Category	Security
Objective	Verify application is protected against XSS attacks
Priority	Critical
Procedure	1. Try to create room with name containing script tag 2. Try to inject JavaScript in Q&A input 3. Check if any user input is rendered unescaped

Expected Result	• Script tags are escaped/sanitized • No JavaScript execution from user input • React's built-in escaping prevents XSS
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** React's default escaping plus DOMPurify on Markdown content blocked all 14 OWASP XSS payloads in our test set.

8.2.3 Reliability and Stability Tests

Table 8.46: TC-REL-001: Frontend Error Display on Backend Unavailability

Test ID	TC-REL-001
Category	Reliability
Objective	Verify the frontend shows a clear error when the backend is unreachable
Priority	Major
Procedure	1. Log in to the application 2. Stop the backend service 3. Attempt to create a new room from the dashboard 4. Observe the error message displayed
Expected Result	• A user-friendly error message is shown (e.g., "Service unavailable") • The application does not crash or show a blank page • No technical stack trace is exposed to the user
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Frontend surfaces a friendly error toast and retries with backoff when /api/backend is down.

Table 8.47: TC-REL-002: API Error Response Format

Test ID	TC-REL-002
Category	Reliability
Objective	Verify backend API returns proper error responses for invalid requests
Priority	Major

Procedure	1. Log in to the application 2. Try to join a room that does not exist by entering a fake room ID in the URL 3. Try to create a room with an empty name 4. Observe the responses
Expected Result	• Non-existent room returns a clear “Room not found” message • Invalid input returns a validation error message • User is not stuck on a broken page
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** 4xx responses follow {detail} schema; 5xx responses include a stable error code and request_id.

Table 8.48: TC-REL-003: Meeting Reconnection After Network Interruption

Test ID	TC-REL-003
Category	Stability
Objective	Verify that a user can recover from a brief network drop during a meeting
Priority	Major
Procedure	1. Join a LiveKit room and confirm connection is established 2. Disconnect from Wi-Fi or network for 5 seconds 3. Reconnect to the network 4. Observe the room UI
Expected Result	• UI shows a “Reconnecting” or connection lost indicator • After network returns, connection re-establishes automatically • Video and audio resume without leaving and rejoining the room
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Failed.** On a network drop > 30 s the LiveKit client does not auto-reconnect on Safari and the user must reload the page. Logged as known-issue ISSUE-217 for v1.1; mitigation: in-UI reconnect button planned.

Table 8.49: TC-REL-004: GPU Server Health Check

Test ID	TC-REL-004

Category	Reliability
Objective	Verify GPU server status endpoint reports system health correctly
Priority	Major
Procedure	1. Ensure the GPU server is running 2. Access the GPU server status endpoint (/status) 3. Process one slide through the system 4. Access the status endpoint again
Expected Result	• Status endpoint returns a successful response with system information • After processing, the slide count is incremented • No error is returned from the status endpoint
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** /status reports patch counts, transcript counts, and CUDA device name; matches DB row counts within 1 s after ingest.

8.2.4 Compatibility Tests

Table 8.50: TC-COMPAT-001: Browser Compatibility

Test ID	TC-COMPAT-001
Category	Compatibility
Objective	Verify application works on Chrome and Firefox
Priority	Major
Procedure	1. Open the application in Google Chrome and complete the full user journey (register, create room, join, share screen, view transcript) 2. Repeat the same steps in Mozilla Firefox
Expected Result	• All features work correctly in both browsers • UI renders consistently across both
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Tested on Chrome 122 / Firefox 124 / Safari 16 — core flows succeed on all three.

Table 8.51: TC-COMPAT-002: Mobile Responsiveness

Test ID	TC-COMPAT-002

Category	Compatibility
Objective	Verify dashboard is viewable on mobile screens
Priority	Minor
Procedure	1. Open the dashboard on a mobile device or use browser dev tools to simulate a mobile screen 2. Check that the room list and navigation are usable
Expected Result	• Dashboard layout adjusts to the smaller screen • Room list and buttons are accessible without horizontal scrolling
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass with notes.** Layout responsive down to 375 px width; on tablet portrait (768 px) the nav collapses but the room card grid keeps a faint horizontal scroll. Tracked as ISSUE-201.

8.2.5 Usability Tests

Table 8.52: TC-USAB-001: First-Time User Onboarding

Test ID	TC-USAB-001
Category	Usability
Objective	Verify a new user can register and join a room without guidance
Priority	Major
Procedure	1. Have a person with no prior knowledge of the system register an account 2. Ask them to create and join a room 3. Observe whether they can complete the tasks without assistance
Expected Result	• User completes all tasks without external help • Error messages (if any) are clear and helpful
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** 5/5 first-time testers completed signup -> create room -> sidebar load within 2 min 40 s avg, no manual consulted.

Table 8.53: TC-USAB-002: Error Message Clarity

Test ID	TC-USAB-002
Category	Usability
Objective	Verify error messages are user-friendly

Priority	Minor
Procedure	1. Attempt to log in with incorrect credentials 2. Attempt to join a non-existent room 3. Read the error messages shown
Expected Result	• Error messages explain what went wrong in plain language • No raw technical errors or stack traces are shown
Date Tested	2026-04-15 – 2026-04-22 (dedicated test sprint)
Result	**Pass.** Error toasts use plain Turkish or English messages without exposing stack traces; UX feedback survey average 4.4/5.

8.3 Test Results Summary

The test sprint ran from 2026-04-15 to 2026-04-22. Functional cases were exercised on the dedicated staging stack (one Linux x86_64 application host running the Compose stack and a separate 24 GB GPU host running the patch-level GPU server). Non-functional and performance cases were measured on the same hardware with no other workload.

8.3.1 Aggregate Results

Table 8.54: Test Outcome Aggregate

Category	Total	Pass	Pass w/ Notes	Failed
Authentication	5	5	0	0
Room Management (Zoom + dashboard)	4	4	0	0
Transcription	3	3	0	0
Slide & Patch Pipeline	5	5	0	0
Question Answering	5	5	0	0
Zoom Integration (incl. RTMS)	4	4	0	0
LiveKit (legacy, single smoke test)	1	1	0	0
Bot WebSocket Delivery	1	1	0	0
GPU Server / Reset / LaTeX	4	4	0	0
Desktop Companion (primary path)	3	3	0	0
Performance	5	4	1	0
Security	5	5	0	0
Reliability	4	3	0	1
Compatibility	2	1	1	0
Usability	2	2	0	0
Maintainability	1	1	0	0
Total	**54**	**51**	**2**	**1**

Strict pass rate: 51/54 = 94.4%. Including “Pass with notes” as a non-blocking pass: 53/54 = 98.1%. Per the team’s “minimum-viable test surface for non-primary paths”

policy, the legacy self-hosted LiveKit room and the alternative Zoom RTMS integration each get exactly one end-to-end smoke test (TC-LK-001 and TC-ZOOM-002 respectively); the rest of the test surface is invested in the primary Recall.ai → Desktop Companion path and the GPU patch-level pipeline. The single hard failure (TC-REL-003: Safari WebRTC reconnection after > 30 s drop) affects only the legacy self-hosted LiveKit path and is tracked as ISSUE-217. The primary desktop-companion path is unaffected: its WebSocket fan-out auto-reconnects within ∼1.4 s (TC-DESK-003).

8.3.2 Performance Headlines

- **Transcript surface latency**: median 1.6 s, p95 2.1 s end-to-end (Deepgram Nova 3 streaming through the bot to the sidebar).
- **Slide ingest**: cold 7.8 s (YOLO + GLM-OCR over ∼8 text patches + bge-m3 + mCLIP); warm-cache 1.4 s when GLM-OCR hits the disk cache.
- **Q&A end-to-end**: median 4.4 s, p95 5.7 s on the reference 24 GB GPU with 5 patch images and `enable_thinking=False`; well within the NFR-PERF-1 budget of 6 s.
- **VRAM usage during `generate()`**: peak 19.8 GB on the reference 24 GB GPU — under the NFR-VRAM-1 cap of 22 GB with ∼4 GB headroom.
- **Retrieval recall@5** (visual stream, mCLIP): 0.91 on a hand-labelled set of 50 Turkish lecture questions; text stream (bge-m3) recall@5: 0.94 on the same set.

8.3.3 Known Issues After Sign-Off

Table 8.55: Open Issues at Final Report Submission

Issue	Description	Severity
ISSUE-217	Safari does not auto-reconnect to LiveKit after a > 30 s network drop. Workaround: page reload. Planned: in-UI manual reconnect for v1.1.	Medium
ISSUE-201	Faint horizontal scrollbar on tablet portrait widths (∼768 px) on the dashboard.	Low
ISSUE-188	Dashboard render time slightly over the 1 s soft target with >50 rooms; pagination planned.	Low
ISSUE-145	Safari 16 occasionally drops the second remote video for ∼1 s after join.	Low

8.3.4 Out-of-Scope

The following classes of testing were explicitly out-of-scope for this academic delivery and are listed for transparency: large-scale concurrent-meeting load tests (>10 simultaneous rooms on the same GPU host), formal penetration testing, regulatory compliance certification (KVKK / GDPR audits), and accessibility certification (WCAG 2.2 AA).

Chapter 9

Maintenance Plan and Details

This chapter sets out how the PARSE system will be kept healthy after the senior-project delivery, through corrective, adaptive, perfective, and preventive maintenance, and how the team intends to operate the system through its first year of academic use.

9.1 Maintenance Categories

Table 9.1: Maintenance Categories and Responses

Category	**What it covers**	**PARSE response**
Corrective	Defects discovered after release (crashes, wrong answers, broken integrations).	Tracked in GitHub Issues; severity-1 patched within 48 h, severity-2 within one week, severity-3 in the next minor release.
Adaptive	Changes in the environment: model API drift (Hugging Face), Zoom SDK updates, LiveKit major versions, GPU driver / CUDA, Postgres major.	Quarterly compatibility sweep; `requirements.txt` pinned with floor versions and a CI run that resolves and tests against latest minors.
Perfective	Improvements requested by users that are not defects: better Turkish phrasing, faster retrieval, lecturer-side dashboard.	Tracked in a separate "enhancement" label; prioritized at the start of each semester.
Preventive	Activities that lower the chance of future defects: refactors, documentation, dependency hygiene, model swaps.	Monthly dependency bump PRs; comparison-notebook reruns every semester to validate model choices.

9.2 Operational Monitoring

The deployed stack is monitored through a small set of always-on signals:

- **Health endpoints**. `/status` on the GPU server reports CUDA device, patch counts, and transcript counts; `/health` on the backend confirms the proxy secret + DB connectivity.

- **Container logs**. Bot and GPU-server logs flow into Docker logging; long-running processes are wrapped in `restart: unless-stopped` in Compose so a crashed process is brought back automatically.
- **Sentry error tracking**. Every PARSE service ships with the Sentry SDK initialized at startup (Python services use `sentry-sdk` with the FastAPI integration; the desktop companion and the Next.js dashboard use `@sentry/electron` and `@sentry/nextjs` respectively). Uncaught exceptions, slow transactions, and frontend JavaScript errors stream to a single Sentry project tagged by `service` and `release` (Git short-SHA). Performance traces are sampled at 10 % to keep ingest cost bounded. Alert rules: any new issue type in production pages the on-call channel; error rate $> 1\,\%$ over 5 min triggers a Slack webhook.
- **GPU telemetry**. `nvidia-smi` sampled every 60 s on the GPU host with VRAM and temperature alarms (Slack webhook on > 22 GB sustained or >80°C).
- **Database growth**. A nightly `psql` job reports row counts per table; thresholds trigger archival.

9.2.1 Sentry Configuration Details

Each service emits a deterministic `release` tag so a regression can be bisected to the exact deploy:

Table 9.2: Sentry SDK Configuration per Service

Service	SDK	Notable settings
GPU Server (FastAPI)	`sentry-sdk[fastapi]`	`traces_sample_rate=0.1`; ignores `HTTPException` 4xx; tags every event with `meeting_id` when present so a Q&A regression can be reproduced from the failing meeting.
Bot (FastAPI)	`sentry-sdk[fastapi]`	Same baseline; additionally captures Recall.ai webhook payload IDs via `set_context` so transcript-fan-out failures are debuggable.
Backend (FastAPI)	`sentry-sdk[fastapi]`	PII scrubbed (Better Auth tokens, X-Proxy-Secret) via `before_send`.
Web Dashboard (Next.js)	`@sentry/nextjs`	Replays disabled by default to respect privacy; user opt-in only.
Desktop Companion (Electron)	`@sentry/electron` (main + renderer)	Native crash reports forwarded to the same project; release matches the dashboard's.

The Sentry DSN is supplied per-environment through a `SENTRY_DSN` environment variable; absent variable disables the SDK (the dev workflow runs without telemetry).

PII rules are enforced through a shared `before_send` hook that strips Better Auth session cookies, X-Proxy-Secret, transcript text bodies, and slide image bytes from any captured payload — only metadata (`meeting_id`, `slide_id`, durations, status codes) is retained.

9.3 Backup and Recovery

- **Postgres**: nightly logical `pg_dump` of `parse_db`, retained for 30 days; a weekly full snapshot retained for 6 months. Restore drill performed once per semester against a staging volume.
- **OCR cache**: `patch_ocr_cache/` is regenerable from raw slides, so it is not in the backup set; documented in the runbook.
- **Model weights**: PARSE never modifies weight files. Re-pulling from Hugging Face is the recovery path; a local mirror is kept on the GPU host so downloads are not needed during a service restart.
- **Recovery objectives**: RPO 24 h (daily backup), RTO 1 h on a fresh host (Docker Compose up + DB restore + model warm).

9.4 Deployment and Rollback

- **Single-command deploy**: `docker compose up -d -build` brings up the application stack on the host. The GPU server is started as a separate `python server.py` process with a systemd unit on the GPU box.
- **Versioning**: each release is a Git tag; the bot, backend, and frontend images are tagged with the same Git short-SHA so they are pinned together.
- **Rollback**: the previous image tag is kept available. Rollback is one command (`docker compose pull && docker compose up -d`) plus a Postgres point-in-time restore from the latest pre-deploy snapshot if a destructive migration was involved.
- **Schema migrations**: `INIT_SQL` is idempotent; the patch-level rewrite tolerates an old 768-dim `slide_patches` by dropping and recreating it. Post-rollout this branch is removed once all environments are on the new schema.

9.5 Dependency and Model Lifecycle

Table 9.3: Dependency and Model Cadence

Component	Cadence	Owner
Frontend npm deps	Monthly minor bump PR	Frontend lead
Backend / bot pip deps	Monthly minor bump PR	Backend lead
`transformers` on GPU server	Tracked separately; bumped at major Qwen/GLM-OCR releases	ML lead
Qwen3.5-4B	Re-evaluated every semester against newer Qwen releases	ML lead
bge-m3	Re-evaluated every semester against multilingual successors	ML lead
mCLIP	Re-evaluated every semester against multilingual CLIP successors	ML lead
Postgres / pgvector	Pinned to current major; major bumps in summer break	DevOps lead
LiveKit (legacy)	Pinned to current major; major bumps in summer break	DevOps lead
Sentry SDKs (Python + JS)	Monthly minor bump PR; release tag updated per deploy	DevOps lead

9.6 Support Channels and SLAs

- **User-facing channel**: an in-app "Help" link opens a contact form that lands in the team's shared inbox. Response target: 24 h on weekdays.
- **Bug reports**: GitHub Issues on the public repository. Severity-1 acknowledgement target: 4 h.
- **Status page**: `status.parse.app` surfaces incident summaries and historical uptime.
- **SLA targets** (after the academic project closes): 99.0% monthly uptime for the dashboard; 98.0% for the GPU server (single-GPU host, expected planned downtime for model upgrades).

9.7 Documentation and Knowledge Transfer

The repository keeps four canonical documents up to date so a future maintainer can take over without team context:

- `CLAUDE.md` — engineering notes covering architecture, ports, env vars, and common workflows.
- `gpu-server/notebooks/` — the comparison and full-pipeline notebooks that justify each ML choice.

- This Final Report — the “why” document.
- `frontend/docs/USER_MANUAL.md` — the embedded user manual referenced from the in-app Help screen.

Chapter 10

Consideration of Various Factors in Engineering Design

This chapter discusses how various factors affected the engineering design process of PARSE, including public health, safety, security, welfare, and global, cultural, social, environmental, and economic factors.

10.1 Constraints

10.1.1 Technical Constraints

1. **GPU Availability**: System requires NVIDIA GPU with CUDA support for ML inference
2. **Network Bandwidth**: Real-time video processing requires stable, high-bandwidth connection
3. **API Rate Limits**: Deepgram and Zoom APIs have usage limits
4. **Browser Compatibility**: WebRTC support varies across browsers

10.1.2 Business Constraints

1. **Development Timeline**: Academic semester constraints
2. **Budget**: Limited to open-source tools and free API tiers where possible
3. **Team Size**: 5-member team with distributed expertise

10.2 Standards

The following standards and best practices were followed:

- **OAuth 2.0**: For Zoom authentication integration
- **JWT (RFC 7519)**: For LiveKit token generation
- **WebRTC**: W3C standard for real-time communication

- **REST API Design**: Following OpenAPI specifications
- **WCAG 2.1**: Accessibility guidelines for UI
- **GDPR**: Data protection considerations for EU users

10.3 Factor Analysis

10.3.1 Public Health

Effect Level: 7/10

PARSE contributes positively to public health by:

- Enabling remote participation, reducing need for physical meetings
- Providing accessibility features (transcription) for hearing-impaired users
- Supporting educational sessions that can include health-related content

Design Decisions:

- Implemented real-time transcription for accessibility
- Designed for remote-first usage patterns
- Ensured low latency to prevent eye strain from delayed video

10.3.2 Safety

Effect Level: 5/10

Safety considerations in PARSE design:

- No physical safety risks (software-only system)
- Cognitive safety through clear UI/UX design
- Data safety through encryption and access controls

Design Decisions:

- Clear visual indicators for recording/sharing status
- Confirmation dialogs for destructive actions
- Session timeout to prevent unauthorized access

10.3.3 Security

Effect Level: 9/10

Security is a critical concern for PARSE:

- Handles sensitive meeting content (video, audio, slides)
- Manages user authentication and sessions
- Processes potentially confidential business presentations

Design Decisions:

- Multi-layer authentication (Better Auth + proxy secret)
- JWT tokens with limited lifetime
- Data isolation between meetings
- HTTPS/WSS for all external communication
- No persistent storage of video/audio by default

10.3.4 Welfare

Effect Level: 6/10

PARSE supports user welfare through:

- Reducing cognitive load with AI-assisted Q&A
- Enabling asynchronous access to meeting content
- Supporting diverse learning styles through multiple modalities

Design Decisions:

- Clean, uncluttered interface
- Optional transcription panel (can be hidden)
- Configurable notification settings

10.3.5 Global Factors

Effect Level: 6/10

Global considerations:

- Users may be distributed across time zones
- Varying internet infrastructure quality globally
- Different data protection regulations by country

Design Decisions:

- Timestamps use UTC with local conversion
- Adaptive bitrate for varying network conditions
- Self-hosted option for data sovereignty

10.3.6 Cultural Factors

Effect Level: 4/10
Cultural considerations:

- UI text in English (primary market)
- Meeting norms vary by culture
- Visual design preferences differ

Design Decisions:

- Neutral, professional color scheme
- Extensible architecture for future localization
- No culturally-specific imagery or icons

10.3.7 Social Factors

Effect Level: 7/10
Social impact of PARSE:

- Enables inclusive participation (accessibility)
- Supports education and knowledge sharing
- May affect presenter-audience dynamics

Design Decisions:

- Transcription visible to all participants equally
- Q&A feature democratizes asking questions
- Speaker identification maintains accountability

10.3.8 Environmental Factors

Effect Level: 5/10
Environmental considerations:

- Server energy consumption for GPU inference
- Reduced travel through remote meetings (positive)
- Electronic device usage

Design Decisions:

- Efficient model inference (batching where possible)
- On-demand GPU usage (not always-on)
- Optimized data transfer to reduce bandwidth

10.3.9 Economic Factors

Effect Level: 8/10
Economic considerations:

- Cost of GPU infrastructure
- API costs (Deepgram, Zoom)
- Potential cost savings for users vs. competitors

Design Decisions:

- Open-source architecture reduces licensing costs
- Efficient transcription (3-second chunks) balances cost/accuracy
- Self-hosted option eliminates recurring SaaS fees

10.4 Factor Summary Table

Table 10.1: Engineering Design Factor Impact Summary

Factor	**Effect (0-10)**	**Primary Impact**
Public Health	7	Accessibility, remote work enablement
Safety	5	Data protection, clear UI indicators
Security	9	Multi-layer auth, data isolation
Welfare	6	Reduced cognitive load, accessibility
Global	6	Time zones, varying infrastructure
Cultural	4	Neutral design, localization-ready
Social	7	Inclusive participation, knowledge sharing
Environmental	5	GPU efficiency, reduced travel
Economic	8	Open-source, efficient API usage

Chapter 11

Ethics and Professional Responsibilities

This chapter discusses the ethical and professional responsibilities the team identified, accepted, and acted on while building and operating PARSE. Because the system processes lecture audio, slides, and student questions — all of them potentially personal data — the ethical posture is not a postscript but a design constraint.

11.1 Privacy and Data Protection

PARSE captures three categories of data on behalf of its users: (1) speech audio from the meeting, transcribed to text via Deepgram Nova 3; (2) slide images shared by the lecturer; and (3) the questions students type into the sidebar. Each of these can be sensitive.

- **Consent**. The bot only joins meetings the user explicitly created or invited it to. The Zoom sidebar is loaded by the student in their own session and never publishes anything back to the lecturer or other participants. The bot is registered as a hidden subscriber on LiveKit so its presence is not displayed.
- **Per-meeting isolation**. Every Postgres query in the GPU server is scoped to `meeting_id` (NFR-SEC-2). One meeting's slides, transcripts, or Q&A history can never surface in a different meeting.
- **Retention and deletion**. `POST /reset` wipes `slide_patches`, `transcripts`, and `qa_history` for a given meeting. Beyond that the team's intent is to follow KVKK / GDPR principles: minimal collection, opt-in retention, and a one-click full delete from the user's account page.
- **Storage hygiene**. No third-party analytics or trackers are embedded in the dashboard; transcripts and slides do not leave the team's infrastructure except for the one round trip to Deepgram for transcription.

11.2 Intellectual Property of Lecture Material

The slides and the spoken content of a lecture are typically owned by the lecturer or the institution. PARSE never republishes either to a third party, never trains a model on

them, and never uses them to generate content outside the meeting they were captured in. The retrieval index is per-meeting and per-user; OCR text and transcript lines exist solely to power that user's own questions about the same lecture.

11.3 Honesty about AI Limitations

A vision-language model can fabricate plausible-sounding answers. PARSE's stance is that an answer is only useful when grounded in what was actually in the lecture:

- The Turkish system prompt explicitly forbids drawing on outside knowledge and requires the model to say "Bilgi verilen slaytlarda bulunmuyor." when grounding is absent.
- Every answer in the UI is rendered together with the source patches it cited, so the user can audit visually where the answer came from.
- The team avoids any framing that suggests PARSE is "smart" or a tutor. Promotional copy and the in-app help text describe PARSE as a classroom companion that recovers what was in the room, not as an oracle.

11.4 Inclusion and Accessibility

PARSE's primary design audience is Turkish-speaking university students, including those who study from recordings rather than live attendance. Concrete accessibility commitments:

- Turkish-first language path everywhere user copy is shown; no English-only screens in core flows.
- Transcripts are searchable text, helping students with hearing loss follow live class.
- KaTeX-rendered math means equations are real text, not images, so screen readers can announce them.
- Color contrast on the dashboard meets WCAG 2.2 AA for body text (verified informally with axe-core).

11.5 Operating Responsibly

- **Security disclosure**. The team operates a documented disclosure address; reports are acknowledged within 48 h.
- **Open-source conduct**. The repository is licensed and the README credits the upstream models (DocLayout-YOLO, GLM-OCR, bge-m3, mCLIP, Qwen3.5-4B). Pull requests follow conventional commits and a code of conduct.
- **Avoiding overreach**. Lecturer-side analytics (planned in Future Work) will be opt-in and aggregated, never exposing individual students.

11.6 Professional Responsibilities Recognized and Fulfilled

Through this project the team explicitly recognized responsibilities listed in the IEEE/ACM Software Engineering Code of Ethics, including: acting consistently with the public interest (Principle 1), being honest about limitations (Principle 7), and supporting colleagues (Principle 6). These were operationalized in the practices above — no dark patterns, no hidden data collection, plain-language disclosure of what the system can and cannot do — and form the standard the team commits to maintaining as PARSE evolves.

Chapter 12

Teamwork Details

This chapter describes how the team collaborated throughout the project, including individual contributions, collaborative environment creation, and leadership sharing.

12.1 Contributing and Functioning Effectively on the Team

12.1.1 Anıl Kılıç – DevOps and Infrastructure Lead

Primary Contributions:

- Designed Docker Compose deployment architecture
- Set up LiveKit server configuration
- Implemented bot service for media processing
- Configured ngrok tunnels for Zoom integration
- Created CI/CD pipelines and deployment scripts

Technical Skills Applied:

- Docker and container orchestration
- LiveKit server administration
- Network configuration and tunneling
- Shell scripting and automation

Effective Functioning:

- Maintained reproducible deployment environments
- Created comprehensive setup documentation
- Supported team with local development issues

12.1.2 Aybars Buğra Aksoy – Backend Development Lead

Primary Contributions:

- Designed and implemented the FastAPI backend
- Created the room management and authentication services
- Integrated LiveKit token generation
- Implemented Zoom OAuth flow and SDK integration
- Set up the database schema for Better Auth

Technical Skills Applied:

- FastAPI and async Python
- PostgreSQL and asyncpg
- OAuth 2.0 implementation
- JWT token handling

Effective Functioning:

- Provided clear API documentation for frontend
- Implemented comprehensive error handling
- Maintained backward compatibility during iterations

12.1.3 Barış Yaycı – ML/AI Integration Lead

Primary Contributions:

- Designed and implemented the GPU server
- Integrated DocLayout-YOLO, GLM-OCR, bge-m3, mCLIP, and Qwen3.5-4B into a patch-level pipeline
- Built the slide processing pipeline
- Implemented the Q&A engine with RAG
- Set up pgvector for semantic search

Technical Skills Applied:

- PyTorch and Transformers
- Computer vision (YOLO, CLIP)
- Vision-language models (Qwen3.5-4B unified VL)
- Vector databases and RAG

Effective Functioning:

- Optimized models for production GPU constraints
- Documented ML pipeline for reproducibility
- Provided benchmarks for performance testing

12.1.4 Bora Yetkin – Frontend Development Lead

Primary Contributions:

- Designed and implemented the Next.js frontend architecture
- Built the dashboard, room management, and authentication pages
- Integrated LiveKit React components for video/audio
- Implemented the transcription panel with real-time updates
- Created responsive UI with TailwindCSS

Technical Skills Applied:

- React 19 and Next.js 16 App Router
- Better Auth integration
- WebSocket and DataChannel handling
- UI/UX design principles

Effective Functioning:

- Maintained clear component documentation
- Responded to integration issues within 24 hours
- Participated in all code reviews

12.1.5 Eren Berk Eraslan – Zoom and AI/ML Integration Lead

Primary Contributions:

- Designed and implemented the GPU server
- Integrated AI pipeline to GPU Server and Zoom SDK
- Implemented Speech to Text transcription
- Designed and implemented the Zoom sidebar React application
- Built Zoom OAuth integration flow and token management
- Implemented RTMS webhook handling and media stream routing
- Developed the end-to-end testing strategy and test case specifications
- Created integration test procedures for cross-service validation

Technical Skills Applied:

- React 18 and Zoom Meeting SDK
- OAuth 2.0 and Zoom API integration

- WebSocket communication and real-time data streaming
- Manual integration testing methodology

Effective Functioning:

- Coordinated Zoom-specific testing across bot and frontend teams
- Maintained Zoom app compliance with marketplace requirements
- Provided test coverage reports and identified critical gaps

12.2 Helping Create a Collaborative and Inclusive Environment

12.2.1 Communication Practices

The team established effective communication channels:

- **Daily Standups**: 15-minute daily sync via Discord
- **Weekly Deep Dives**: 1-hour technical discussions
- **Shared Documentation**: Confluence wiki for decisions
- **Code Reviews**: All PRs required 1+ approval

12.2.2 Knowledge Sharing

Team members actively shared knowledge:

- Bora Yetkin conducted React hooks workshop for the team
- Aybars Buğra Aksoy documented API patterns for frontend integration
- Barış Yaycı presented ML model selection rationale
- Anıl Kılıç created deployment runbooks
- Eren Berk Eraslan organized Zoom SDK integration sessions and testing workshops

12.2.3 Inclusive Practices

The team ensured all voices were heard:

- Rotating meeting facilitator role
- Anonymous feedback surveys after milestones
- Pair programming sessions across expertise areas
- Asynchronous decision-making for flexibility

12.2.4 Conflict Resolution

When disagreements arose, the team:

- Discussed technical trade-offs objectively
- Used prototyping to validate competing approaches
- Escalated to advisor when consensus couldn't be reached
- Documented decisions and rationale

12.3 Taking Lead Role and Sharing Leadership on the Team

12.3.1 Distributed Leadership Model

Each member led specific project areas:

Table 12.1: Leadership Distribution

Member	Lead Role	Leadership Activities
Anıl Kılıç	DevOps Lead	Infrastructure decisions, deployment strategy, security configuration
Aybars Buğra Aksoy	Backend Lead	API design, database schema, integration coordination
Barış Yaycı	ML Lead	Model selection, inference optimization, accuracy benchmarking
Bora Yetkin	Frontend Lead	UI/UX decisions, component architecture, user experience review
Eren Berk Eraslan	Zoom & Testing Lead	Zoom integration, RTMS setup, test strategy, quality assurance

12.3.2 Leadership Rotation

For cross-cutting concerns, leadership rotated:

- **Sprint Planning**: Rotated every 2-week sprint
- **Demo Presentations**: Each member presented their component
- **Documentation Review**: Weekly rotation
- **Integration Testing**: Collaborative, Aybars Buğra Aksoy coordinated

12.3.3 Decision-Making Authority

Leaders had authority within their domain:

- Technology choices within component (e.g., Frontend Lead chose TailwindCSS)
- Implementation details without team vote
- Code review approval for their area

Cross-domain decisions required consensus:

- API contract changes
- Data model modifications
- Security policy changes

12.3.4 Leadership Development

Each member grew their leadership skills:

- Bora Yetkin: Improved stakeholder communication
- Aybars Buğra Aksoy: Enhanced technical documentation
- Barış Yaycı: Developed cross-functional collaboration
- Anıl Kılıç: Strengthened project coordination
- Eren Berk Eraslan: Built quality assurance leadership

12.4 Meeting Objectives

This section reports the project objectives that were originally captured in the Analysis and Requirements Report's project plan, against the state at final delivery. Objectives are grouped by milestone in the original plan; "Met", "Partially Met", and "Deferred" are used as outcome labels.

Table 12.2: Project Plan Milestones — Final Status

Milestone	Planned Objective	Status	Evidence
M1 – Project setup	Repository, infrastructure (Docker Compose, Postgres, LiveKit), CI baseline.	Met	`docker-compose.yml`, GitHub Actions, repository structure (Section 5.1).
M2 – Auth + room CRUD	Better Auth, room create/list/join/delete via FastAPI backend.	Met	TC-AUTH-* and TC-ROOM-* in Chapter 8.

Table 12.2: Project Plan Milestones — Final Status

Milestone	Planned Objective	Status	Evidence
M3 – Live media	Bot subscribes to a meeting (Recall.ai-driven primary path; legacy LiveKit subscriber retained).	Met	TC-DESK-002, TC-BOT-002, TC-LK-001.
M4 – Real-time transcription	Deepgram Nova 3 integration with sidebar broadcast.	Met	TC-TRANS-* and TC-PERF-001.
M5 – Slide ingest	DocLayout-YOLO patching + per-patch storage in pgvector.	Met (rev.)	Original plan called for whole-slide CLIP; replaced with the patch-level pipeline (TC-SLIDE-*).
M6 – Vision-language Q&A	VLM-driven answers grounded in slides and transcripts.	Met (rev.)	Replaced Qwen2.5-VL with Qwen3.5-4B + GLM-OCR + bge-m3 + mCLIP for Turkish quality.
M7 – Zoom integration	Sidebar app inside Zoom + RTMS audio/desk-share + transcript correlation.	Met	TC-ZOOM-*, TC-BOT-002, TC-TRANS-002.
M8 – Desktop companion	Native macOS capture, drawing, notes.	Met	TC-DESK-001 and the desktop-design branch merged into main.
M9 – Documentation & User Manual	In-app help, README, and final report.	Met	In-app Help screen, `frontend/docs/USER_MANUAL.md`, this report.
M10 – Performance & VRAM budget	Q&A under 6 s on a 24 GB GPU; transcript surface under 2 s.	Met	NFR-PERF-1 (median 4.4 s), NFR-PERF-3 (median 1.6 s); see Section 8.3.
M11 – Lecturer dashboard	Anonymized view for instructors of which slides students asked about.	Deferred	Moved to Future Work (Section 14.2).
M12 – Multi-language UI	UI translations beyond Turkish/English.	Deferred	Out of scope; recorded as Future Work.

10 of the 12 planned milestones were met by final delivery (two of them with revised technical content versus the original Analysis report — the slide-ingest and Q&A pipelines were upgraded mid-project to deliver Turkish quality the original choices could not). The two deferred items (lecturer dashboard, multi-language UI) are tracked as Future Work and were openly de-prioritized in favor of investing the available time in the patch-level

retrieval pipeline that materially raised end-user answer quality.

Chapter 13

New Knowledge Acquired and Applied

This chapter records the knowledge each team member acquired in service of the project and the strategies used to acquire it. Many of these areas were outside the standard CS curriculum at Bilkent, so they required structured self-study, reproducible experimentation, and peer teaching inside the team.

13.1 Per-Member Knowledge Acquisition

Anıl Kılıç (DevOps and Infrastructure Lead)

Acquired: Docker Compose multi-service orchestration, LiveKit self-hosting (TURN/STUN, SFU configuration, port mapping), `platform: linux/amd64` cross-architecture builds for Apple Silicon dev environments, Postgres with the `pgvector` extension, GPU host operations (`nvidia-smi`, CUDA/PyTorch matching, model warmup). Strategies: official LiveKit and pgvector docs, DigitalOcean and AWS deployment guides, hands-on staging environment kept identical to production.

Aybars Buğra Aksoy (Backend Development Lead)

Acquired: FastAPI async patterns, `asyncpg` connection-pool management, Better Auth schema and JWT signing for LiveKit, Zoom OAuth + Meeting SDK signature generation, robust proxy-secret middleware design. Strategies: Better Auth source code reading, FastAPI advanced documentation, Zoom developer portal, paired implementation reviews with the frontend lead.

Barış Yaycı (ML/AI Integration Lead)

Acquired: vision-language models (Qwen2.5-VL, Qwen3-VL-4B, Qwen3.5-4B), document layout detection (DocLayout-YOLO), specialized OCR systems (GLM-OCR, PaddleOCR-VL, LightOnOCR), multilingual embeddings (bge-m3, mCLIP variants), pgvector hybrid retrieval, fp16/bf16 mixed-precision serving on a 24 GB budget. Strategies: research papers (CLIP, BGE, DocLayout-YOLO), Hugging Face model cards, hands-on comparison notebooks (`compare_ocr_and_text_embeddings.ipynb`,

`compare_patch_embeddings.ipynb`), and step-by-step ablations on real Turkish lecture slides.

Bora Yetkin (Frontend Development Lead)

Acquired: Next.js 16 App Router patterns, React 19 server actions, Tailwind v4 theming with CSS variables, Better Auth client-side hooks, LiveKit React components, Excalidraw embedding inside a custom canvas, KaTeX-based math rendering. Strategies: Next.js docs, the Vercel deployment guides, design references like Linear's changelog and Anthropic's documentation site, peer code reviews on every PR.

Eren Berk Eraslan (Zoom and ML/AI Integration Lead)

Acquired: Zoom RTMS SDK (callback threading model, webhook signatures), Recall.ai bot integrations as an alternative to native RTMS, perceptual-hash slide change detection, transcript–slide correlation strategies, Deepgram Nova 3 streaming. Strategies: Zoom Developer documentation, Recall.ai onboarding, hands-on debugging against recorded Zoom sessions, instrumentation of slide-change false-positive rates.

13.2 Cross-Team Practices

- **Reproducible notebooks**. All ML model decisions are accompanied by a notebook in `gpu-server/notebooks/` that loads the candidates, runs them on the same Turkish lecture corpus, and prints the comparison table. New team members can re-run the notebooks to verify the choices.
- **Pair-implementation**. The two largest pipelines (Zoom RTMS handler and the patch-level GPU server) were implemented in pair sessions across two leads to spread tacit knowledge.
- **Documentation-as-code**. Engineering decisions live in `CLAUDE.md` alongside the source so they are reviewed in the same PR, not in a separate wiki that drifts.
- **Branch experiments**. Big refactors (the Deepgram correlation, the patch-level pipeline, the desktop redesign) lived on branches long enough to be reviewed end-to-end before being merged, which kept `main` stable through the academic year.

13.3 Learning Strategies Applied

- **Read the paper, then build the smallest version**. For every model we adopted, we read at least the abstract and method section of the original paper before writing integration code, then reproduced the result on a tiny test set before introducing it into the system.
- **Compare in writing**. For OCR and CLIP variants we wrote per-pair comparisons in a notebook (cells labelled by the candidate's name) before discussing the choice in a team review — this kept conversations grounded in evidence rather than preference.

- **Steal from the platforms**. Wherever a platform vendor (Zoom, LiveKit, Recall.ai, Hugging Face) provided sample apps, we cloned them and modified incrementally rather than reinventing.
- **Teach across the team**. Each lead ran a short internal walkthrough of their subsystem after a major milestone — this kept the system collectively understood, not siloed.

Chapter 14

Conclusion and Future Work

14.1 Conclusion

PARSE delivers a complete, working classroom companion for Turkish-speaking university students attending online or hybrid lectures. The system attends every minute of class, watches every slide, hears every spoken sentence, and keeps the link between the two so a student who steps away for thirty seconds can recover the moment with a question. The implementation runs end-to-end on commodity infrastructure (a Linux application host plus a single 24 GB GPU) and answers Turkish questions in under five seconds, grounded in the actual lecture rather than in generic model knowledge.

The team's most consequential design decision during the project was the move from the original whole-slide CLIP-plus-Qwen2.5-VL baseline to the patch-level pipeline (DocLayout-YOLO + GLM-OCR + bge-m3 + mCLIP + Qwen3.5-4B). That decision was not in the initial plan, was made against direct comparison evidence in the notebooks, and is the single biggest reason the final system is usable on Turkish lectures rather than merely demonstrable.

14.1.1 What Was Delivered

- A Next.js dashboard, FastAPI backend, dual-mode bot, GPU server, Zoom sidebar, and Electron desktop companion — six services that compose into a single product surface for the student.
- Slide-correlated transcripts, OCR-grounded patch embeddings, and a Turkish-first Q&A path with regenerate, drawing notes, and KaTeX math support.
- A repository of ablation notebooks that justify the model choices and serve as living documentation.
- A Final Report, an embedded user manual, and a maintenance plan that together let a future maintainer run the system without team context.

14.1.2 Lessons Learned

- **Multilingual quality is a product feature, not a postscript.** An English-first pipeline silently degraded everywhere on Turkish input; only after we replaced the

whole stack with multilingual-from-the-start models did the product feel like it was speaking the user's language.

- **Disk caches earn their keep.** The GLM-OCR per-patch cache cut re-ingest latency 5x and made iterative testing of the rest of the stack practical.
- **Grounding is the trustworthy default.** Every time we relaxed the "answer only from the lecture" constraint during development the model produced confident wrong answers; tightening the system prompt was always the right move.
- **Sequence the integrations.** Doing Zoom RTMS in parallel with LiveKit nearly destabilized the bot; serializing the integration milestones (M3 then M7) saved weeks.

14.1.3 Limitations

- Single-GPU deployment caps concurrent meeting count; horizontal scale for the GPU server was not implemented.
- Safari WebRTC reconnection on > 30 s drops requires manual reload (ISSUE-217).
- Lecturer-side analytics, multi-language UI, and semester-spanning memory are roadmap, not delivered.
- All retrieval recall numbers reported in this document are measured on a single Turkish lecture corpus; broader-curriculum evaluation is pending.

14.2 Future Work

14.2.1 Lecturer Dashboard

An anonymized view for instructors showing which slides drew the most student questions and which OCR'd terms were most queried — giving lecturers a feedback signal they have never had access to in real time. To be added without weakening per-student privacy: aggregates only, with a minimum cohort threshold.

14.2.2 Streaming Generation

Switching the GPU server to streamed token generation will let answers appear as they form rather than after the full `generate()` returns, cutting the user's perceived latency by roughly half on the longer answers.

14.2.3 Voice Question Input

Allowing students to whisper their question to PARSE while the lecturer is mid-sentence — faster than typing, still silent in the room. Will reuse the existing Deepgram integration.

14.2.4 Semester Memory

Indexing across an entire semester of lectures so a question can be answered from the relevant past class even when the student is in a different live session. Requires extending pgvector retrieval to a multi-meeting scope and adding consent UI for cross-meeting use.

14.2.5 Lecturer-Side Slide Quality Linting

A pre-class screen that runs the same DocLayout-YOLO + GLM-OCR pass on a lecturer's slide deck and flags slides where the OCR is unreliable (low contrast, decorative fonts, scanned PDFs), so the lecturer can fix the source before the slides reach students.

14.2.6 Open-Source Model Fine-Tuning

A small Turkish-specific fine-tune of Qwen3.5-4B using lecturer-approved Q&A pairs and slide content, to push answer quality further without sacrificing groundedness.

Chapter 15

Glossary

API (Application Programming Interface)
A set of protocols and tools for building software applications and enabling communication between different software components.

Better Auth
An open-source authentication library for JavaScript/TypeScript applications, used in PARSE for user session management.

Bot In PARSE context, an automated service that joins meetings to process audio/video streams.

CLIP (Contrastive Language-Image Pre-training)
OpenAI's neural network model that learns visual concepts from natural language supervision, used for generating image embeddings.

CUDA (Compute Unified Device Architecture)
NVIDIA's parallel computing platform and programming model for GPU computing.

DataChannel
WebRTC feature allowing peer-to-peer data transfer, used in PARSE for transmitting transcriptions.

Deepgram
A speech recognition platform providing STT (Speech-to-Text) services, specifically the Nova 3 model used in PARSE.

Docker
A platform for developing, shipping, and running applications in containers.

Embedding
A numerical representation of data (text or images) in a high-dimensional vector space.

FastAPI
A modern, fast Python web framework for building APIs.

GPU (Graphics Processing Unit)
A specialized processor designed for parallel processing, used for ML model inference.

JWT (JSON Web Token)
A compact, URL-safe means of representing claims to be transferred between two parties.

LiveKit
An open-source, self-hosted WebRTC platform for building real-time video and audio applications.

Next.js
A React framework for building full-stack web applications.

OAuth 2.0
An authorization framework enabling third-party applications to obtain limited access to a web service.

pgvector
PostgreSQL extension for storing and querying vector embeddings.

Qwen3.5-4B
Alibaba's unified multilingual vision-language model, used for grounded Turkish Q&A generation in PARSE.

GLM-OCR
An open vision-language model from Zhipu AI optimized for document text recognition; PARSE invokes it per YOLO patch with the prompt "Text Recognition:".

bge-m3
BAAI's multilingual text-embedding model (1024-dim) used to index OCR'd text patches for retrieval.

mCLIP
Multilingual CLIP variant (`xlm-roberta-base-ViT-B-32`) used to embed non-text patch crops in 512-dim image space.

RAG (Retrieval-Augmented Generation)
A technique combining information retrieval with text generation for improved AI responses.

RTMS (Real-Time Media Streams)
Zoom's API for accessing raw audio and video streams from meetings.

STT (Speech-to-Text)
The process of converting spoken language into written text.

Transcription
The written form of spoken content, generated in real-time during meetings.

VLM (Vision Language Model)
An AI model that can process and understand both visual and textual information.

WebRTC (Web Real-Time Communication)
A technology enabling peer-to-peer audio, video, and data sharing in web browsers.

WebSocket
A protocol providing full-duplex communication channels over a single TCP connection.

YOLO (You Only Look Once)
A real-time object detection algorithm, specifically DocLayout YOLO for document analysis.

Chapter 16

References

1. Bruegge, B., & Dutoit, A. H. (2004). *Object-Oriented Software Engineering: Using UML, Patterns, and Java* (2nd ed.). Prentice Hall.
2. Next.js Documentation. (2024). Retrieved from https://nextjs.org/docs
3. FastAPI Documentation. (2024). Retrieved from https://fastapi.tiangolo.com/
4. LiveKit Documentation. (2024). Retrieved from https://docs.livekit.io/
5. Zoom Developer Documentation. (2024). RTMS API. Retrieved from https://developers.zoom.us/docs/api/rtms/
6. Deepgram Documentation. (2024). Nova 3 Model. Retrieved from https://developers.deepgram.com/
7. pgvector GitHub Repository. (2024). Retrieved from https://github.com/pgvector/pgvector
8. Ultralytics YOLO Documentation. (2024). Retrieved from https://docs.ultralytics.com/
9. OpenCLIP GitHub Repository. (2024). Retrieved from https://github.com/mlfoundations/open_clip
10. BAAI bge-m3 Model Card. (2024). Hugging Face. Retrieved from https://huggingface.co/BAAI/bge-m3
11. GLM-OCR Model Card. Zhipu AI. (2024). Hugging Face. Retrieved from https://huggingface.co/zai-org/GLM-OCR
12. Qwen3.5-4B Model Card. Alibaba Cloud. (2025). Hugging Face. Retrieved from https://huggingface.co/Qwen/Qwen3.5-4B
13. Recall.ai Documentation. (2025). Retrieved from https://docs.recall.ai
14. Deepgram Nova 3 Documentation. (2025). Retrieved from https://developers.deepgram.com/docs/nova-3

15. Qwen2.5-VL Model Card. (2024). Hugging Face. Retrieved from https://huggingface.co/Qwen/Qwen2.5-VL (initial baseline; superseded by Qwen3.5-4B in the final pipeline)

16. Better Auth Documentation. (2024). Retrieved from https://www.better-auth.com/docs

17. PostgreSQL Documentation. (2024). Retrieved from https://www.postgresql.org/docs/

18. Docker Documentation. (2024). Retrieved from https://docs.docker.com/

19. OAuth 2.0 Authorization Framework. RFC 6749. Retrieved from https://tools.ietf.org/html/rfc6749

20. JSON Web Token (JWT). RFC 7519. Retrieved from https://tools.ietf.org/html/rfc7519

21. WebRTC 1.0: Real-Time Communication Between Browsers. W3C. Retrieved from https://www.w3.org/TR/webrtc/

22. OWASP Top Ten. (2021). Retrieved from https://owasp.org/www-project-top-ten/

23. Software Testing Help. (2024). Manual Testing Tutorial. Retrieved from https://www.softwaretestinghelp.com/manual-testing-tutorial-1/