

# EDITH — Complete Technical Architecture Report

**EDITH** — Elite Digital Intelligence & Task Handler A Chrome Extension-based AI agent with autonomous browser automation, multi-tab parallel research, and intelligent document processing.

## Table of Contents

- [1. Technology Stack](#)
- [2. High-Level System Architecture](#)
- [3. Extension Architecture — The Core Agent](#)
- [4. Modes of Operation](#)
- [5. Data Flow](#)
- [6. Browser Automation Engine](#)
- [7. Multi-Tab Research Orchestrator](#)
- [8. Tool Ecosystem](#)
- [9. Document Intelligence Module](#)
- [10. Storage & State Management](#)
- [11. LLM Integration](#)
- [12. Project File Structure](#)
- [13. Security Architecture](#)

## 1. Technology Stack

### 1.1 Chrome Extension — Core Agent

Technology	Version	Purpose
WXT	0.20.17	Chrome extension framework (Manifest V3)
TypeScript	5.7.3	Type-safe logic across all modules
React	18.3.1	Sidepanel & New Tab UI
React DOM	18.3.1	DOM rendering for extension pages
OpenAI SDK	4.86.2	LLM API calls directly from browser context
TailwindCSS	3.4.17	Utility-first styling for extension UI
PostCSS	8.5.3	CSS processing pipeline
Chrome Debugger API	CDP	DOM automation via Chrome DevTools Protocol

**Chrome Extension Permissions:** debugger · sidePanel · storage · tabs · activeTab · scripting · alarms · notifications · <all\_urls>

### 1.2 Document Intelligence Module

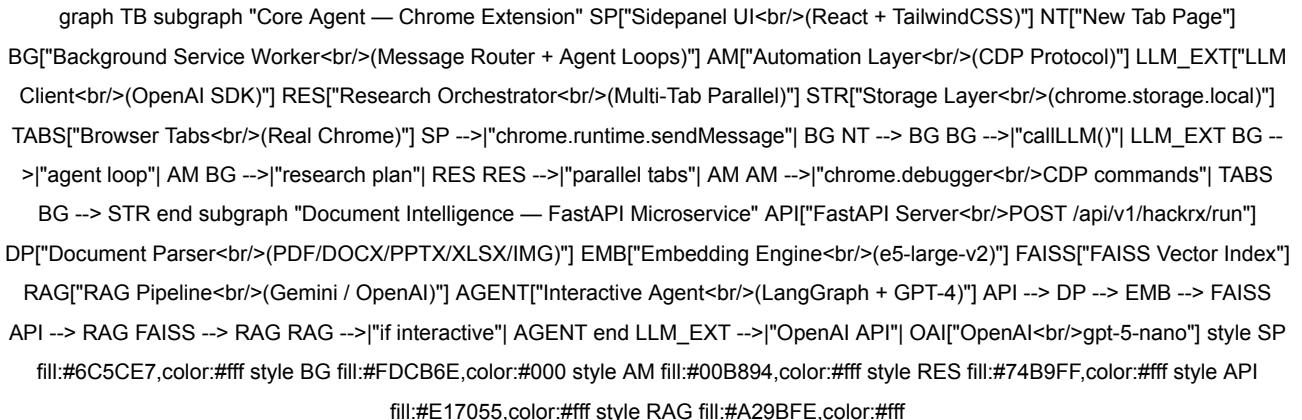
Technology	Purpose
<b>FastAPI + Uvicorn</b>	REST API server
<b>Sentence-Transformers</b> ( <code>intfloat/e5-large-v2</code> )	Text embedding model
<b>FAISS (CPU)</b>	Vector similarity search
<b>PyMuPDF (fitz)</b>	PDF parsing with hyperlink extraction
<b>python-docx</b>	Word document parsing
<b>python-pptx</b>	PowerPoint parsing + embedded image OCR
<b>OpenPyXL + Pandas</b>	Excel/CSV parsing & analysis
<b>Pillow + Pytesseract</b>	OCR for images and embedded visuals
<b>Google GenAI</b> ( <code>gemini-2.5-flash</code> )	Primary LLM for Q&A
<b>OpenAI</b> ( <code>gpt-5-nano</code> )	Fallback LLM
<b>LangChain + LangGraph</b>	Interactive reasoning agent orchestration
<b>BeautifulSoup4</b>	Web scraping for linked documents

## 1.3 External AI Services

Service	Model	Role
<b>OpenAI</b>	<code>gpt-5-nano</code> (configurable)	Primary agent LLM — reasoning, tool-calling, intent
<b>OpenAI</b>	<code>gpt-4</code>	Interactive reasoning agent (Document Intelligence)
<b>Google Gemini</b>	<code>gemini-2.5-flash</code>	Primary LLM for document Q&A (with key rotation)

# 2. High-Level System Architecture

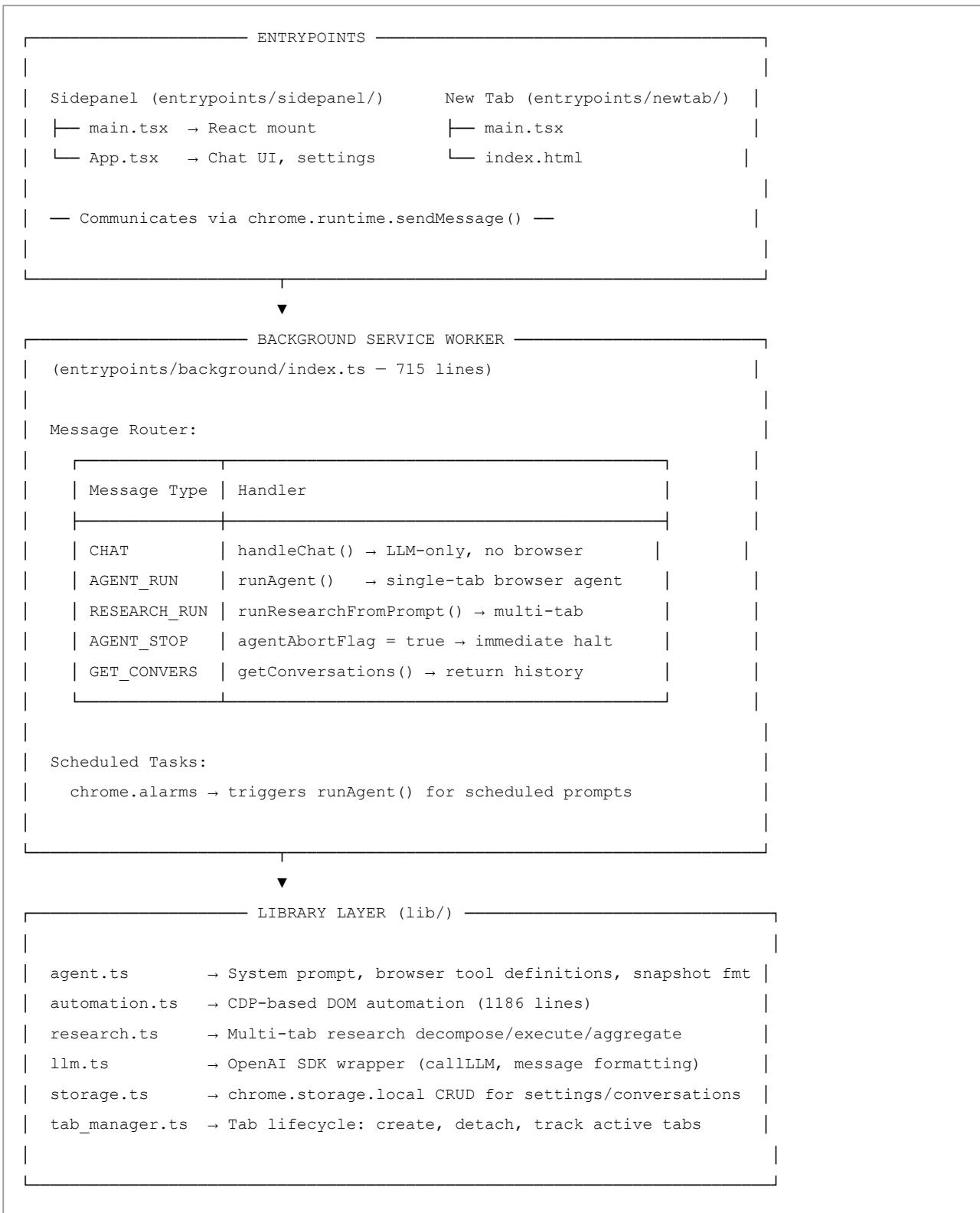
EDITH is composed of **two independent systems** that together form the project:



# 3. Extension Architecture — The Core Agent

The extension follows a **3-layer architecture**: UI → Background Service Worker → Automation Layer.

## 3.1 Layer Breakdown



### 3.2 Key Source Files & Responsibilities

File	Lines	Role
<a href="#">background/index.ts (file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH- main/extension/background/index.ts)</a>	715	Service worker: message routing, agent loop, research runner, alarm handler

File	Lines	Role
<a href="#">agent.ts (file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH- main/extension/lib/agent.ts)</a>	398	System prompt (140 lines), BROWSER_TOOLS definitions (15 tools), <code>formatSnapshot()</code> . <code>(file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH- main/extension/lib/agent.ts#141-242),</code> <code>pruneHistory()(file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH- main/extension/lib/agent.ts#243-262)</code>
<a href="#">automation.ts (file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH- main/extension/lib/automation.ts)</a>	1186	CDP automation: <code>takeSnapshot()</code> (JS injection), <code>clickElement()</code> , <code>typeText()</code> , <code>pressKey()</code> , <code>scrollPage()</code> , <code>selectOption()</code> , <code>hoverElement()</code> , <code>setValue()</code> <code>decomposeTask()(file:///d:/EDITH%20-%20Final%20yr%20project/EDITH- main/EDITH- main/extension/lib/research.ts#95-138),</code> <code>runSubTask()(file:///d:/EDITH%20-%20Final%20yr%20project/EDITH- main/EDITH- main/extension/lib/research.ts#190-482),</code> <code>aggregateResults()(file:///d:/EDITH%20-%20Final%20yr%20project/EDITH- main/EDITH- main/extension/lib/research.ts#495-523)</code> — MapReduce-style parallel research
<a href="#">research.ts (file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH- main/extension/lib/research.ts)</a>	523	
<a href="#">llm.ts (file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH- main/extension/lib/llm.ts)</a>	99	OpenAI SDK client, message format conversion, tool schema mapping
<a href="#">storage.ts (file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH- main/extension/lib/storage.ts)</a>	134	Typed CRUD for settings, conversations (100 max), MCP servers, scheduled tasks
<a href="#">tab_manager.ts (file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH- main/extension/lib/tab_manager.ts)</a>	~100	<code>createTab()</code> , <code>detachAll()</code> , multi-tab lifecycle for research

## 4. Modes of Operation

Every user message enters the background service worker and is routed to one of **four execution modes**:

### 4.1 CHAT Mode — Simple Conversation

```
sequenceDiagram participant U as User (Sidepanel) participant BG as Background Worker participant LLM as OpenAI API U->>BG: { type: "CHAT", prompt: "Hello!" } BG->>BG: Load settings + conversation BG->>LLM: callLLM(prompt, history, tools=[]) Note over BG,LLM: No browser tools provided<br/>Pure text conversation LLM-->>BG: "Hi! I can help you browse..." BG->>BG: Save to chrome.storage.local BG-->>U: sendResponse({ ok: true })
```

#### Characteristics:

- **No browser tools** are passed to the LLM (empty tools array)
- Synchronous response — sidepanel waits for `sendResponse`
- Conversation persisted to `chrome.storage.local`

- Used for: greetings, questions about capabilities, general chat
- 

## 4.2 AGENT\_RUN Mode — Single-Tab Browser Automation

sequenceDiagram participant U as User (Sidepanel) participant BG as Background Worker participant LLM as OpenAI API participant CDP as Chrome Debugger participant TAB as Browser Tab U->>BG: { type: "AGENT\_RUN", prompt: "Search Amazon for laptop" } BG->>U: { ok: true } (immediate ack) Note over U: Sidepanel now listens<br/>for broadcastEvent() loop Agent Loop (max 30 steps) BG->>LLM: callLLM(SYSTEM\_PROMPT, history, BROWSER\_TOOLS) alt LLM returns tool\_calls LLM-->>BG: [{ name: "open\_browser", args: {url: "amazon.com"} }] BG-->>BG: progress("open\_browser: ...") BG->>CDP: openBrowser("https://amazon.com") CDP->>TAB: chrome.debugger.attach + Page.navigate CDP-->>BG: tabId BG-->>BG: sleep(1500ms) Note over BG: Next iteration... else LLM returns tool: take\_snapshot BG-->>CDP: takeSnapshot(tabId) CDP-->>TAB: Runtime.evaluate(SNAPSHOT\_JS) TAB-->>CDP: { url, title, elements[], rawText } CDP-->>BG: PageSnapshot BG-->>BG: formatSnapshot(snapshot) Note over BG: Elements formatted with UIDs<br/>Prioritized: inputs → buttons → links else LLM returns tool: click(uid) BG->>CDP: clickElement(uid, snapshot, tabId) CDP-->>TAB: Input.dispatchEvent(x, y) BG->>BG: sleep(1200ms) BG-->>CDP: auto-takeSnapshot() Note over BG: Auto-snapshot after every action else LLM returns task\_complete LLM-->>BG: { name: "task\_complete", args: {summary: "Done!"} } BG-->>BG: Save final message BG-->>CDP: detachDebugger(tabId) BG-->>U: broadcastEvent({ type: "agent\_done" }) Note over BG: EXIT agent loop else LLM returns text (no tools) LLM-->>BG: "Task completed." BG-->>U: broadcastEvent({ type: "agent\_done" }) Note over BG: EXIT agent loop end end

### Characteristics:

- Asynchronous** — sends immediate ack, then broadcasts progress events
  - Max 30 steps** per run
  - 15 browser tools** provided via `BROWSER_TOOLS` from `agent.ts` ([file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH-main/extension/lib/agent.ts](#))
  - Auto-snapshot** after every mutating action (click, type\_text, press\_key, hover, select\_option, set\_value)
  - Snapshot loop detection** — warns after 3 consecutive snapshots without action
  - New tab detection** — automatically switches `activeTabId` when clicks open new tabs
  - Abort support** — `agentAbortFlag` checked before each iteration and each tool execution
- 

## 4.3 RESEARCH\_RUN Mode — Multi-Tab Parallel Research

sequenceDiagram participant U as User (Sidepanel) participant BG as Background Worker participant LLM as OpenAI API participant TM as Tab Manager participant T1 as Tab 1 participant T2 as Tab 2 participant T3 as Tab 3 U->>BG: { type: "RESEARCH\_RUN", prompt: "Compare iPhone 16 prices" } BG-->>U: { ok: true } (immediate ack) rect rgb(240, 248, 255) Note over BG,LLM: Phase 1: Task Decomposition BG-->>LLM: decomposeTask(prompt) LLM-->>BG: { isResearch: true, subTasks: [<br/> {url: "amazon.com", goal: "iPhone 16 price"}, <br/> {url: "flipkart.com", goal: "iPhone 16 price"}, <br/> {url: "apple.com", goal: "official price"}] } end rect rgb(255, 248, 240) Note over BG,T3: Phase 2: Parallel Execution BG-->>TM: createTab(amazon.com) → tabId1 BG-->>TM: createTab(flipkart.com) → tabId2 BG-->>TM: createTab(apple.com) → tabId3 BG-->>BG: sleep(2000ms) par Run all tabs simultaneously BG-->>T1: runSubTask(settings, subTask1, tabId1) Note over T1: Agent loop: snapshot → type → search → extract and BG-->>T2: runSubTask(settings, subTask2, tabId2) Note over T2: Agent loop: snapshot → type → search → extract and BG-->>T3: runSubTask(settings, subTask3, tabId3) Note over T3: Agent loop: snapshot → type → search → extract end end rect rgb(240, 255, 240) Note over BG,LLM: Phase 3: Result Aggregation BG-->>LLM: aggregateResults(prompt, subTaskResults[]) LLM-->>BG: "iPhone 16 prices:\n• Amazon: ₹79,900\n• Flipkart: ₹78,999\n• Apple: ₹79,900" end BG-->>TM: detachAll() BG-->>U: broadcastEvent({ type: "agent\_done" })

### Characteristics:

- MapReduce pattern** — decompose → parallel execute → synthesize
- Max 5 concurrent tabs** (configurable via `MAX_RESEARCH_TABS`)

- Each sub-task tab runs its own independent agent loop with `extract_data` as the terminal tool (instead of `task_complete`)
  - **Minimum 2 sub-tasks** required — otherwise falls back to Agent mode
  - Uses `Promise.allSettled()` — partial failures don't kill the entire research
  - Tabs remain open after research so the user can manually review sources
- 

## 4.4 AGENT\_STOP Mode — Emergency Halt

```
sequenceDiagram participant U as User participant BG as Background Worker participant LOOP as Agent/Research Loop
U->>BG: { type: "AGENT_STOP" }
BG->>BG: agentAbortFlag = true
BG-->U: { ok: true }
Note over LOOP: Next iteration checks flag LOOP->>LOOP: if (agentAbortFlag) break
LOOP->>BG: detachDebugger()
BG->>U: broadcastEvent("agent_done")
```

### Characteristics:

- Sets a global `agentAbortFlag` boolean
  - Checked at the **top of each agent iteration** and **before each tool execution**
  - Gracefully saves conversation state before stopping
  - Detaches CDP debugger to free browser resources
- 

# 5. Data Flow

## 5.1 End-to-End Agent Flow

```
flowchart TD
    A["User types in Sidepanel"] --> B["chrome.runtime.sendMessage()"]
    B --> C["Message Router<br/>(background/index.ts)"]
    C -->|{"type: CHAT"}| D["handleChat()"]
    C -->|{"type: AGENT_RUN"}| E["runAgent()"]
    C -->|{"type: RESEARCH_RUN"}| F["runResearchFromPrompt()"]
    C -->|{"type: AGENT_STOP"}| G["Set abort flag"]
    D --> H["callLLM(prompt, [], no tools)"]
    H --> I["Save response → chrome.storage"]
    I --> J["sendResponse to sidepanel"]
    E --> K["callLLM(SYSTEM_PROMPT, <br/>pruneHistory(messages), <br/>BROWSER_TOOLS)"]
    K --> L["LLM Response"]
    L --> M["tool_calls"]
    M --> N["Execute via CDP"]
    N --> O["Append result to history"]
    O --> P["task_complete"]
    P --> Q["decomposeTask(prompt)"]
    Q --> R["tabManager.createTab() × N"]
    R --> S["Promise.allSettled(<br/>runSubTask per tab)"]
    S --> T["aggregateResults(results)"]
    T --> P
    style C fill:#FDDB6E,color:#000
    style M fill:#00B894,color:#fff
    style K fill:#6C5CE7,color:#fff
    style Q fill:#74B9FF,color:#fff
```

## 5.2 Snapshot Lifecycle

The snapshot is the **central data structure** that bridges the LLM and the browser:

```
flowchart LR
    A["Chrome Tab<br/>(real webpage)"] -->|"chrome.debugger<br/>Runtime.evaluate"| B["SNAPSHOT_JS<br/>(injected script)"]
    B -->|"Scans DOM"|
    C["PageSnapshot"]
    C --> D["formatSnapshot()"]
    D --> E["Prioritized Text<br/>for LLM"]
    E -->|"Sent as tool result"|
    F["LLM"]
    F -->|"Returns: click(uid: 42)"|
    G["clickElement(42, snapshot)"]
    G -->|"Lookup uid → x,y coords"|
    H["CDP: Input.dispatchEvent"]
    H --> A
    subgraph PageSnapshot [PageSnapshot]
        C1["url: string"]
        C2["title: string"]
        C3["elements: SnapshotElement[]"]
        C4["rawText: string (5000 chars)"]
    end
    subgraph SnapshotElement [SnapshotElement]
        C1["uid: number"]
        E1["tag, role, name"]
        E2["x, y, width, height"]
        E3["context (parent section)"]
        E5["value, placeholder"]
        E6["checked, disabled, options[]"]
    end
```

### Snapshot processing pipeline:

1. **Inject** `SNAPSHOT_JS` into the page via CDP `Runtime.evaluate`
2. **Scan** all DOM elements — filter by clickable tags, actionable ARIA roles, and input elements
3. **Classify** each element's type: INPUT, BUTTON, LINK, CHECKBOX, SELECT, RADIO, etc.
4. **Assign UIDs** — sequential numeric IDs based on position

5. **Extract context** — walk up to 5 parent levels for `aria-label`, `id(file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH-main/backend/app/services/mcp_service.py#1582-1594)`, `class hints`
  6. **Prioritize** — Tier 1 (inputs) → Tier 2 (buttons/checkboxes) → Tier 3 (product cards) → Tier 4 (links)
  7. **Format** as a flat text string with `uid | TYPE | "label"` [in: section-context]
  8. **Append** the page's `rawText` (first 5000 chars) for context
- 

## 6. Browser Automation Engine (CDP)

The automation layer in [automation.ts \(file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH-main/extension/lib/automation.ts\)](#) uses the **Chrome DevTools Protocol** via `chrome.debugger`:

### 6.1 CDP Architecture

```
graph TD
    A["automation.ts"] --> B["ensureAttached(tabId)"]
    B --> C["cdp(tabId, method, params)"]
    C --> D["chrome.debugger.attach(tabId, '1.3')"]
    D --> E["chrome.debugger.sendCommand(tabId, method, params)"]
    E --> F["Page.navigate"]
    E --> G["Runtime.evaluate"]
    E --> H["Input.dispatchMouseEvent"]
    E --> I["Input.dispatchKeyEvent"]
    E --> J["DOM.getDocument"]
    E --> K["Page.captureScreenshot"]
    E --> L["Network events"]

    A --> D
    A --> C
    A --> E
    A --> F
    A --> G
    A --> H
    A --> I
    A --> J
    A --> K
    A --> L
```

### 6.2 Key Automation Functions

Function	CDP Method	What It Does
<code>openBrowser(url)</code>	<code>chrome.tabs.create + ensureAttached(file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH-main/extension/lib/automation.ts#43-58)</code>	Opens new tab, attaches debugger, navigates
<code>navigateTo(url)</code>	<code>Page.navigate</code>	In-tab navigation
<code>takeSnapshot()</code>	<code>Runtime.evaluate(SNAPSHOT_JS)</code>	Injects 300-line JS to scan interactive DOM elements
<code>clickElement(uid)</code>	<code>Input.dispatchMouseEvent</code>	Maps UID → (x,y) from snapshot, dispatches mouse events
<code>typeText(text, uid)</code>	<code>Runtime.evaluate(focus) + Input.dispatchKeyEvent</code>	Character-by-character typing with 30-80ms delays
<code>pressKey(key)</code>	<code>Input.dispatchKeyEvent (keyDown + keyUp)</code>	Simulates Enter, Tab, Escape, arrows
<code>scrollPage(dir)</code>	<code>Runtime.evaluate(window.scrollBy)</code>	Scroll up/down/top/bottom
<code>selectOption(uid, value)</code>	<code>Runtime.evaluate</code>	Programmatically sets select value + fires change event

Function	CDP Method	What It Does
hoverElement(uid)	Input.dispatchMouseEvent (mouseMoved)	Triggers hover states for dropdown menus
setValue(uid, value)	Runtime.evaluate	Direct .value injection for numeric fields
takeScreenshot()	Page.captureScreenshot	PNG screenshot of visible viewport
waitForNetworkIdle()	Network event monitoring	Waits until no pending network requests
<u><a href="#">waitForLoad()(file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH-main/extension/lib/automation.ts#82-100)</a></u>	Page.loadEventFired	Waits for page load completion
detachDebugger()	chrome.debugger.detach	Releases debugger from tab

## 6.3 Multi-Tab Support

The automation layer tracks attached tabs via `attachedTabs: Set<number>`:

- Multiple debugger sessions can be active simultaneously (for research)
- `lastSingleTabId` provides backward compatibility for single-tab agent runs
- `detachAllDebuggers()` cleans up all sessions at once

# 7. Multi-Tab Research Orchestrator

Defined in [research.ts \(file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH-main/extension/lib/research.ts\)](#), the research system uses a **3-phase MapReduce** architecture:

## 7.1 Phase 1: Task Decomposition (decomposeTask) ([file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH-main/extension/lib/research.ts#95-138](#))

An LLM call analyzes the user's prompt and decides:

- **Is this a research task?** (`isResearch: boolean`)
- **What sub-tasks are needed?** Each with: `description, url (file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH-main/backend/app/services/mcp_service.py#1082-1088), extractionGoal`

The LLM is given strict criteria for what counts as research:

*Genuine research requiring 2+ different websites — NOT simple browsing, messaging, or single-site tasks.*

## 7.2 Phase 2: Parallel Execution ([runSubTask \(file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH-main/extension/lib/research.ts#190-482\)](#))

Each sub-task gets:

- Its own Chrome tab via `tabManager.createTab(url)`
- Its own agent loop (similar to [runAgent \(file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH-main/extension/entrypoints/background/index.ts#141-495\)](#) but with `extract_data` as terminal tool)
- Its own LLM conversation context
- An abort signal connection to the global abort flag

**Sub-task tools** are the same as `BROWSER_TOOLS` but with `task_complete` replaced by `extract_data` (which captures findings instead of ending the run).

## 7.3 Phase 3: Aggregation ([aggregateResults \(file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH-main/extension/lib/research.ts#495-523\)](#))

An LLM call receives all extracted data and the original prompt, then synthesizes:

- Key findings from each source
- Differences and similarities across sources
- Source citations for every data point
- Tables or bullet points for comparisons

---

# 8. Tool Ecosystem

## 8.1 Browser Tools (defined in [agent.ts \(file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH-main/extension/lib/agent.ts\)](#))

Tool	Description	Auto-Snapshots?
<code>task_complete</code>	Signals task completion — <b>stops the agent immediately</b>	—
<code>open_browser</code>	Opens URL in new tab	No (manual snapshot needed)
<code>take_snapshot</code>	Gets all interactive elements with UIDs	—
<code>click</code>	Clicks element by UID	<input type="checkbox"/> Auto-snapshot after

Tool	Description	Auto-Snapshots?
type_text	Types text char-by-char into field by UID	<input type="checkbox"/> Auto-snapshot after
press_key	Keyboard events (Enter, Tab, Escape, etc.)	<input type="checkbox"/> Auto-snapshot after
scroll	Scrolls page up/down	No
<u>screenshot (file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH-main/backend/app/services/mcp_service.py#1113-1119)</u>	Captures visual screenshot	No
select_option	Selects dropdown option by value	<input type="checkbox"/> Auto-snapshot after
hover	Triggers hover/mouseover on element	<input type="checkbox"/> Auto-snapshot after
set_value	Direct value injection for inputs	<input type="checkbox"/> Auto-snapshot after
wait_for_page_update	Waits for AJAX/network idle	<input type="checkbox"/> Auto-snapshot after
<u>navigate (file:///d:/EDITH%20-%20Final%20yr%20project/EDITH-main/EDITH-main/extension/lib/automation.ts#68-81)</u>	In-tab URL navigation	No

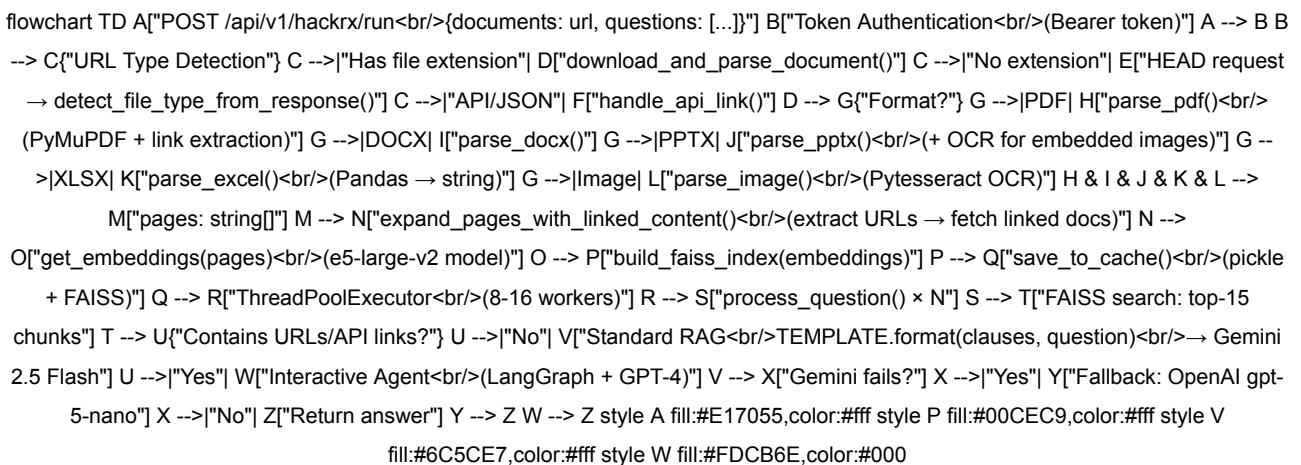
## 8.2 Research-Only Tools

Tool	Description
extract_data	Terminal tool for research sub-tasks — captures extracted information

# 9. Document Intelligence Module

A standalone FastAPI microservice for intelligent document Q&A, located at `Document Intelligence/doc_intel/`.

## 9.1 Architecture



## 9.2 Dual-Path Processing

The system dynamically chooses between two processing paths:

### Path A: Standard RAG Pipeline

For regular documents without interactive instructions:

1. Parse document → chunks
2. Embed with e5-large-v2 → FAISS index
3. For each question: semantic search → top-15 chunks → LLM prompt
4. Gemini 2.5 Flash (primary) → OpenAI (fallback)

### Path B: Interactive Reasoning Agent

For documents containing URLs, API endpoints, or multi-step instructions:

1. Detects that relevant chunks contain URLs/API references via `contains_api_or_url()`
2. Activates the **LangGraph ReAct agent** with two tools:
  - `document_retriever` — searches the loaded document via FAISS
  - `web_scraper_tool` — fetches data from external URLs/APIs
3. The agent reasons through multi-step instructions autonomously

## 9.3 Key Features

Feature	Implementation
<b>Multi-format support</b>	PDF, DOCX, PPTX, XLSX/XLS, JPG/PNG/GIF (OCR)
<b>Linked document expansion</b>	Auto-extracts URLs from document text, fetches and indexes linked content
<b>Embedding caching</b>	MD5-hashed filenames in <code>pdf_cache/</code> — persistent across requests
<b>Concurrent Q&amp;A</b>	<code>ThreadPoolExecutor</code> with 8-16 workers for batch question processing
<b>LLM key rotation</b>	3 Gemini API keys cycled via <code>itertools.cycle()</code>
<b>OneDrive/SharePoint</b>	Auto-converts sharing URLs to direct download URLs
<b>OCR for images in PPTX</b>	Extracts embedded images from slides and runs Pytesseract
<b>Prompt engineering</b>	Domain-specific template for insurance, legal, HR documents

## 9.4 Document Intelligence File Structure

```

Document Intelligence/doc_intel/
├── app/
│   ├── main.py          # FastAPI app, token auth, request logging
│   ├── document_parser.py # parse_pdf, parse_docx, parse_pptx, parse_excel, parse_image
│   ├── embeddings.py    # SentenceTransformer (e5-large-v2) + FAISS index builder
│   ├── retrieval.py     # RAG pipeline, caching, linked doc expansion, question processing
│   ├── interactive_agent.py # LangGraph ReAct agent with document_retriever + web_scraper tools
│   ├── prompt_template.py # Domain-specific RAG prompt template
│   └── utils.py         # clean_response(), contains_api_or_url()
├── pdf_cache/          # Persistent embedding cache (pickle + FAISS)
├── requirements.txt    # 23 Python dependencies
├── Dockerfile           # Docker containerization support
└── .env                 # Gemini keys (x3), OpenAI key, Groq key

```

## 10. Storage & State Management

### 10.1 Extension Storage (`chrome.storage.local`)

```

graph LR
    subgraph "chrome.storage.local"
        A["edith_api_key<br/>(OpenAI API key)"]
        B["edith_api_base_url<br/>(default: api.openai.com/v1)"]
        C["edith_model<br/>(default: gpt-4o-mini)"]
        D["edith_conversations<br/>(last 100 conversations)"]
        E["edith_mcp_servers<br/>(MCP server configs)"]
        F["edith_schedules<br/>(scheduled tasks)"]
    end

```

### 10.2 Data Structures

```

interface Conversation {
  id: string;           // UUID
  title: string;        // First 60 chars of prompt
  messages: Message[]; // Full message history
  createdAt: number;
  updatedAt: number;
}

interface Message {
  id: string;
  role: 'user' | 'assistant' | 'tool';
  content: string;
  toolCalls?: ToolCall[]; // For assistant messages with tool calls
  toolCallId?: string;    // For tool response messages
  toolName?: string;
  timestamp: number;
}

interface ScheduledTask {
  id: string;
  name: string;
  prompt: string;          // Agent prompt to execute
  cronExpression: string;
  enabled: boolean;
  lastRun?: number;
}

```

## 10.3 Document Intelligence Caching

- **In-memory cache:** `pdf_cache` dict (keyed by document URL)
- **Persistent cache:** `pdf_cache/` directory with pickled pages + embeddings
- **Cache key:** MD5 hash of the document URL
- **FAISS index:** Rebuilt from cached embeddings on load (fast)

# 11. LLM Integration

## 11.1 Extension LLM Client

The extension calls OpenAI-compatible APIs directly from the browser via the [OpenAI SDK](#):

```

const client = new OpenAI({
  apiKey: settings.apiKey,
  baseURL: settings.apiBaseUrl,           // User-configurable
  dangerouslyAllowBrowser: true,          // Required for extension context
});

const response = await client.chat.completions.create({
  model: settings.model,                  // User-configurable (default: gpt-4o-mini)
  messages: [system, ...history],
  tools: browserTools,
  tool_choice: 'auto',
  max_completion_tokens: 4096,
});

```

## 11.2 System Prompt Architecture

The `SYSTEM_PROMPT` in `agent.ts` (140 lines) contains:

- **Workflow instructions:** open → snapshot → interact → verify → complete
- **Snapshot reading guide:** How to interpret UIDs, types, and section context
- **Site-specific intelligence:** YouTube, WhatsApp, Gmail, Amazon, e-commerce patterns
- **Forbidden actions:** Never re-open sites, never click navigation links when searching
- **Completion criteria:** URL-based verification for each task type

## 11.3 Conversation History Pruning

`pruneHistory()` keeps only the **last 6 tool exchange rounds** to manage token costs:

- User/assistant text messages are always kept
- Only the most recent tool call/response pairs are retained

## 11.4 Document Intelligence LLM Strategy

```

Question → FAISS top-15 chunks → prompt template

Primary: Gemini 2.5 Flash (3 API keys rotated)
  ↓ on failure
Fallback: OpenAI gpt-5-nano
  ↓ on failure
Return: "Unable to process this query"

If interactive (URLs/APIs detected):
  → LangGraph Agent with OpenAI GPT-4

```

# 12. Project File Structure

```

EDITH - Final yr project/

|
|   └── EDITH-main/EDITH-main/
|       └── extension/
|           ├── wxt.config.ts
|           ├── package.json
|           ├── tsconfig.json
|           ├── tailwind.config.js
|           ├── postcss.config.js
|           └── entrypoints/
|               ├── background/
|               |   └── index.ts
|               ├── sidepanel/
|               |   ├── main.tsx
|               |   ├── App.tsx
|               |   └── index.html
|               └── newtab/
|                   ├── main.tsx
|                   └── index.html
|           └── lib/
|               ├── agent.ts
|               ├── automation.ts
|               ├── research.ts
|               ├── llm.ts
|               ├── storage.ts
|               └── tab_manager.ts
|           └── assets/
|               └── global.css
|
└── Document Intelligence/
    └── doc_intel/
        └── app/
            ├── main.py
            ├── document_parser.py
            ├── embeddings.py
            ├── retrieval.py
            ├── intractive_agent.py
            ├── prompt_template.py
            └── utils.py
        └── pdf_cache/

```

★ CORE AGENT

# Manifest V3 config + permissions  
# Dependencies (OpenAI, React, WXT, TS)  
# TypeScript configuration  
# TailwindCSS theme  
# PostCSS plugins

# ★ Service worker (715 lines)  
# Message router, agent loop,  
# research runner, alarm handler

# React mount  
# Chat UI + settings

# New tab page

# ★ System prompt (140 lines) +  
# BROWSER\_TOOLS definitions +  
# formatSnapshot() + pruneHistory()  
# ★ CDP automation (1186 lines)  
# Snapshot JS, click, type, scroll,  
# hover, select, multi-tab attach  
# ★ Research orchestrator (523 lines)  
# decomposeTask, runSubTask,  
# aggregateResults  
# OpenAI SDK wrapper (99 lines)  
# chrome.storage CRUD (134 lines)  
# Multi-tab lifecycle (~100 lines)

★ DOCUMENT Q&A MODULE

# FastAPI app + auth middleware  
# PDF/DOCX/PPTX/XLSX/Image parsers  
# e5-large-v2 + FAISS builder  
# RAG pipeline + caching + linked docs  
# LangGraph ReAct agent  
# Domain-specific prompt template  
# Utilities  
# Persistent embedding cache

```

├── requirements.txt          # 23 dependencies
├── Dockerfile                # Docker support
└── .env                      # API keys (Gemini ×3, OpenAI, Groq)

```

## 13. Security Architecture

Layer	Mechanism	Details
<b>Extension Permissions</b>	Manifest V3	Scoped: debugger, sidePanel, storage, tabs, scripting, alarms
<b>API Key Storage</b>	chrome.storage.local	User's OpenAI key stored locally, never transmitted to any server except OpenAI
<b>LLM Calls</b>	Direct browser → OpenAI	No intermediary — extension calls API directly with dangerouslyAllowBrowser: true
<b>Agent Safety</b>	Abort flag	User can immediately stop any running agent via AGENT_STOP
<b>Snapshot Loop Prevention</b>	Counter + warning	Detects 3+ consecutive snapshots without action
<b>New Tab Tracking</b>	activeTabId switch	Automatically follows cross-tab navigation
<b>CDP Cleanup</b>	finally block	detachDebugger() always called, even on errors
<b>Doc Intel Auth</b>	Bearer token	Static token validation on all API requests
<b>Gemini Key Rotation</b>	itertools.cycle()	3 keys rotated to avoid rate limits
<b>Conversation Limits</b>	100 max conversations	Prevents unbounded storage growth
<b>History Pruning</b>	Last 6 tool rounds	Prevents token overflow on long agent runs

*Report generated from exhaustive source code analysis of every file in the extension and Document Intelligence modules. All architecture diagrams and data flows are derived directly from the actual implementation.*