Rio

Overview & Vision

Status: Draft v1.0, Work in progress **Last Updated:** November 2025

Github: https://github.com/bhi5hmaraj/rio/tree/main

Executive Summary

Rio is an open-source Chrome Extension that acts as a "Radar Intercept Officer" (RIO/RSO) for AI conversations. While the user (the Pilot) flies the conversation in ChatGPT or other AI interfaces, Rio sits in the back seat (the Chrome Side Panel), actively scanning the chat for hallucinations, bias, and missed nuances.

Rio is a Chrome extension that analyzes web pages and chat conversations in real-time, extracting concepts to build a **Concept DAG** (Directed Acyclic Graph) rendered in a persistent **side-panel HUD**. The HUD hosts a React app with **CopilotKit** (for agent actions) and **React Flow** (for graph visualization).

Unlike passive tools, Rio is **agentic**:

- Scrapes conversations in real-time
- Cross-references claims using Google Search (via Gemini)
- Highlights debatable text directly in the chat interface
- Visualizes conversation structure as an interactive graph
- Provides AI-powered analysis and annotations

Rio operates on a **"Bring Your Own Key" (BYOK)** model for the core extension, ensuring user privacy and zero infrastructure costs. An **optional backend server** (open source, self-hostable) provides advanced features like long-term storage, RAG on conversation history, and proactive analysis across all websites.

Problem Statement

Large Language Models (LLMs) like ChatGPT are powerful but prone to:

1. **Hallucinations:** Stating falsehoods confidently
2. **Sycophancy:** Agreeing with the user even when the user is wrong
3. **Bias:** Non-neutral perspectives that go unnoticed
4. **Complexity:** Long conversations become difficult to track mentally
5. **Lost Context:** Important concepts and relationships get buried in conversation flow

Existing solutions are either:

- Fully separate chat apps (disconnecting you from your workflow)
- Simple overlay scripts that break due to Content Security Policies (CSP) and DOM fragility
- Server-dependent tools that compromise privacy and require infrastructure

Core Value Propositions

1. Real-Time AI Critique

- Analyzes AI responses for logical flaws, factual errors, and bias
- Uses Google Search grounding for fact-checking
- Highlights problematic text directly in the interface

2. Concept Visualization

- Extracts key concepts from conversations
- Maps relationships as an interactive DAG
- Enables mental model building and conversation navigation

3. Privacy-First Architecture

- **BYOK (Bring Your Own Key):** Users provide their own API keys
- **Local-First:** Extension works fully standalone
- **No Analytics:** Zero tracking or telemetry
- **Optional Backend:** Self-hostable server for advanced features (storage, RAG)
- **User-Controlled Data:** Choose between local-only or server sync

4. Robust & Non-Invasive

- Uses Chrome Side Panel (immune to page CSP/Trusted-Types)
- Hypothesis-style text anchoring (survives DOM changes)
- Works across ChatGPT, Gemini, and other AI interfaces

Goals & Non-Goals

Goals

- **Modular Architecture:** Composable components that can be swapped/upgraded independently
- **Robust Anchoring:** Text highlighting that survives DOM drift (quote + position + fuzzy matching)
- **Local-First, Cloud-Optional:** Extension works standalone; backend optional for advanced features
- **Great UX:** Side panel with zoomable DAG, export (SVG/JSON), and Copilot chat
- **Privacy & Transparency:** Open source (extension + backend), client-side processing, user-controlled API keys
- **Cross-Platform:** Works on ChatGPT, Gemini, Claude, and generic web pages
- **Scalable Storage:** Long-term annotation storage via optional self-hosted backend
- **RAG-Enabled:** Query conversation history with natural language (backend feature)

Non-Goals

- **Forking Entire Annotation UIs:** Not rebuilding Hypothesis sidebar; Side Panel is our UI surface
- **Complex Page Injection:** No injecting complex UI into hostile pages (CSP issues)
- **Providing Hosted LLM Services:** Users bring their own API keys (BYOK)
- **Mandatory Backend Dependency:** Extension must work fully offline/standalone
- **Real-Time Collaboration:** v1 focuses on single-user analysis (multi-user in v2)
- **Mobile Support:** Chrome Extension desktop only (for now)

Target Users

Primary

- **Power Users of AI:** People who have extended, complex conversations with ChatGPT/Claude
- **Researchers & Analysts:** Those who need to track concepts across long AI interactions
- **Critical Thinkers:** Users who want to verify AI claims and spot bias

Secondary

- **Developers:** Building on top of Rio's architecture for custom analysis
- **Educators:** Teaching critical thinking with AI tools
- **Privacy Advocates:** Users who want client-side AI tooling

Success Metrics

Adoption

- Chrome Web Store installations
- GitHub stars and community engagement
- Active users (measured via opt-in telemetry if added later)

Utility

- Average highlights per conversation
- DAG exports per session
- User retention (return usage after 7 days)

Quality

- Anchor resolution success rate (>95%)
- False positive rate for hallucination detection
- User-reported bugs vs. features

Architecture

Status: Draft v1.0 **Last Updated:** November 2025

System Overview

Rio is built as a **Manifest V3 Chrome Extension** to bypass CSP limitations and enable a rich UI via the Side Panel API. The architecture follows a "Hybrid" component model with three distinct contexts communicating via the Chrome Runtime API.

The "Hybrid" Component Model

Components & Responsibilities

Component	Role	Runtime Context	Tech Stack	Key Responsibilities
Content Script	"The Hands"	Injected into web page	Vanilla TS + `@hypothesis/text-quote-selector`	• Scrape chat text • Tag DOM elements with stable IDs • Paint colored highlights on page • Render tooltips on hover
Side Panel	"The Face"	Extension page (chrome-extension://)	React + CopilotKit + React Flow	• Main UI/HUD • Display Concept DAG • "Run Critique" triggers • Manage user settings (API Key)
Background Service Worker	"The Brain"	Extension background	Service Worker (TS)	• Orchestrate API calls to Gemini • Handle `chrome.storage` encryption/decryption • Manage global events • Cross-origin fetch (via host_permissions)
Backend Server (Optional)	"The Memory"	Self-hosted server	FastAPI + PostgreSQL + Vector DB	• Long-term annotation storage • RAG on conversation history • Proactive analysis queue • Graph clustering & ML features

Why This Architecture?

1. **Side Panel Isolation**
   - Runs in extension context, immune to page CSP/Trusted-Types
   - Allows React, external scripts, and iframes
   - Persistent UI that doesn't interfere with page layout

- See: Chrome Side Panel API

2. **Content Script Limitations**
   - Can read/modify DOM but inherits page CSP
   - Cannot use `innerHTML` on Gemini (TrustedHTML enforcement)
   - Cannot load external scripts on ChatGPT (CSP blocks)
   - Should be kept minimal and focused on DOM operations only
3. **Background Worker Power**
   - Can make cross-origin fetches (via `host_permissions`)
   - Persistent storage access
   - Can coordinate between multiple tabs/panels
   - Service Worker lifecycle (event-driven, not always running)
4. **Optional Backend Server**
   - Extension works fully standalone (local-first)
   - Backend adds: unlimited storage, RAG, proactive analysis
   - Open source, self-hostable (no vendor lock-in)
   - See: Backend Server Design

Data Flow

The "Critique Loop" (Primary Workflow)

```
??????????????
?  User       ?
?  (clicks    ?
?  "Critique")?
??????????????
       ?
       ?
?????????????????????????
?  Side Panel (React) ?
?  - CopilotKit UI    ?
?????????????????????????
       ? chrome.runtime.sendMessage({action: "critique"})
       ?
?????????????????????????
?  Background Worker   ?
?  - Routes request    ?
?????????????????????????
       ? chrome.tabs.sendMessage({action: "scrape"})
       ?
?????????????????????????
?  Content Script      ?
?  - Scrape chat DOM   ?
?  - Extract messages  ?
?????????????????????????
       ? returns {messages: [...]}
       ?
```

```
??????????????????????????
?  Background Worker   ?
?  - Call Gemini API   ?
?  - With Google Search?
??????????????????????????
       ? Gemini response: {annotations: [...]}
       ?
????????????????????????????????????????????
?  Background broadcasts to:          ?
?  1. Side Panel (for DAG)            ?
?  2. Content Script (for highlights) ?
????????????????????????????????????????????
```

Message Schemas

See Data Models for detailed schemas.

Content → Background (Scrape Result)

```
{
  action: "scrapeComplete",
  data: {
    pageId: string,
    url: string,
    messages: Array<{
      id: string,
      role: "user" | "assistant",
      text: string,
      html: string,
      timestamp: number
    }>
  }
}
```

Background → Side Panel (Analysis Result)

```
{
  action: "analysisComplete",
  data: {
    dag: {
      nodes: Node[],
      edges: Edge[]
    },
    annotations: Annotation[],
    status: "success" | "error",
    error?: string
  }
}
```

Background → Content Script (Highlight Command)

```
{
  action: "applyHighlights",
  annotations: Array<{
    id: string,
    target: {
      messageId: string,
      selector: TextQuoteSelector | TextPositionSelector
    },
    color: "blue" | "green" | "orange" | "red",
    category: "critique" | "factuality" | "sycophancy" | "bias",
    note: string
  }>
}
```

Manifest V3 Configuration

Required Permissions (Minimal Scope)

```
{
  "permissions": [
    "sidePanel",      // For the UI
    "storage",        // For API keys and settings
    "activeTab",      // Minimize warnings; only active when clicked
    "scripting"       // To inject content script
  ],
  "host_permissions": [
    "https://generativelanguage.googleapis.com/*",  // Gemini API
    "https://chat.openai.com/*",                    // ChatGPT scraping
    "https://gemini.google.com/*"                   // Gemini scraping
  ],
  "optional_permissions": [
    "http://localhost:*/*"  // For local development/testing
  ]
}
```

Content Security Policy

The Side Panel (as an extension page) has relaxed CSP and can:

- Load external scripts (React, CopilotKit)
- Use `eval()` if needed (though we avoid it)
- Create iframes
- Use inline scripts

The Content Script inherits the page's CSP and **cannot**:

- Use `innerHTML` on pages with Trusted Types (Gemini)

- Load external scripts on pages with strict CSP (ChatGPT)
- Execute inline scripts

Key Modules (Swappable Components)

1. Scraper (Content Script)

Interface:

```
interface Scraper {
  scrape(): Promise<ScrapedData>;
  detectSite(): "chatgpt" | "gemini" | "claude" | "generic";
}
```

Implementations:

- `ChatGPTScraper`: Uses `[data-message-id]` selectors
- `GeminiScraper`: Uses `.model-response-text` selectors
- `ClaudeScraper`: TBD
- `GenericScraper`: Fallback for articles (readability.js)

Output: Linearized text + DOM map (offsets ↔ nodes)

2. AnchorEngine (Content Script)

Built on Hypothesis standards + libraries.

Interface:

```
interface AnchorEngine {
  createSelector(range: Range): TextQuoteSelector & TextPositionSelector;
  resolveSelector(selector: Selector): Range | null;
}
```

Libraries:

- `@hypothesis/dom-anchor-text-quote`
- `@hypothesis/dom-anchor-text-position`

Features:

- Fuzzy anchoring with context matching

- Falls back to position hints if quote fails
- Uses W3C Web Annotation Data Model

See Text Anchoring for details.

3. AnalyzerAdapter (Background Worker)

Interface:

```
interface AnalyzerAdapter {
  analyze(text: string, options: AnalysisOptions): Promise<AnalysisResult>;
}
```

Implementations:

- `GeminiAnalyzer`: Uses Google Gemini 2.5 Flash with Search grounding
- `LocalMockAnalyzer`: Deterministic, no network (for testing)
- `RemoteLLMAnalyzer`: Custom backend (future)

Output: Normalized `{nodes, edges, annotations}`

4. DAGRenderer (Side Panel)

Interface:

```
interface DAGRenderer {
  render(dag: Graph): void;
  export(format: "svg" | "png" | "json"): Blob;
}
```

Implementations:

- `ReactFlowRenderer`: Interactive, live editing (default)
- `MermaidRenderer`: Static SVG fallback (low-end devices)

5. CopilotLayer (Side Panel)

Integration: CopilotKit hooks

Actions:

- `analyzeCurrentPage`: Triggers the critique loop
- `summarizeSelection`: User highlights text, asks for summary

- `addAnnotation`: Manual annotation creation
- `exportGraph`: Download DAG as file

See UI/UX Design for details.

Security Boundaries

What Content Script CAN Do

 Read page DOM (text, structure) Create temporary overlays (highlights, tooltips) Tag elements with `data-*` attributes Communicate with Background via messages

What Content Script CANNOT Do

 Inject complex HTML (CSP/Trusted Types blocks it) Load external libraries (CSP blocks `<script src>`) Make cross-origin fetches directly Access `chrome.storage` directly (must go through Background)

What Side Panel CAN Do

 Full React app with external dependencies Direct access to `chrome.storage` iframe embedding (if needed) WebGL/Canvas rendering (React Flow)

What Background Worker CAN Do

 Cross-origin fetches (via `host_permissions`) Long-lived operations (within service worker limits) Global state management Tab coordination

Performance Considerations

Content Script

- **Keep it light:** Minimal bundle size (use tree-shaking)
- **Lazy inject:** Only inject when Side Panel is opened
- **Debounce DOM reads:** Use IntersectionObserver for visible content only

- **Highlight batching:** Group DOM updates to avoid layout thrashing

Side Panel

- **Code splitting:** Load React Flow only when Graph tab is active
- **Virtualization:** For large annotation lists (react-window)
- **Memoization:** React.memo for DAG nodes to prevent re-renders

Background Worker

- **Cache API responses:** Use `chrome.storage` for recent analyses
- **Request deduplication:** Don't re-analyze unchanged content
- **Timeout handling:** Abort fetch if Gemini takes >30s

Testing Strategy

Unit Tests

- AnchorEngine: Selector creation/resolution
- Scrapers: DOM extraction logic
- AnalyzerAdapter: API contract compliance

Integration Tests

- Message passing between components
- Storage encryption/decryption
- API error handling

E2E Tests (Playwright)

- Full critique loop on mocked ChatGPT page
- Highlight anchoring accuracy
- DAG rendering

Revision #2
Created 20 March 2026 15:49:56 by bhishma
Updated 20 March 2026 15:54:27 by bhishma