Product Insights

Table of Contents

1. Architecture Overview 2
    - Architecture Summary 2
        - Assumptions 2
        - Core Persona 3
2. High-Level Architecture Diagram 4
3. Component Breakdown 5
    - 3.1 Cloudflare Pages (Frontend Hosting) 5
        - Role of the Component 5
        - Competitive Benchmarking 5
        - Why Cloudflare Pages Was Chosen 5
        - Enhancement Opportunities 6
    - 3.2 Cloudflare Workers (Core Execution Layer) 6
        - Role of the Component 6
        - Competitive Benchmarking 6
        - Why Cloudflare Workers Were Chosen 7
        - Enhancement Opportunities 7
    - 3.3 Cloudflare D1 (Structured Storage) 8
        - Role of the Component 8
        - Competitive Benchmarking 8
        - Why Cloudflare D1 Was Chosen 8
        - Enhancement Opportunities 9
    - 3.4 Cloudflare Workers AI (AI Insights) 9
        - Role of the Component 9
        - Competitive Benchmarking 9
        - Why Cloudflare Workers AI Was Chosen 10
        - Enhancement Opportunities 10
    - 3.5 Cloudflare K,V Cache (Model response and data refresh) 11
        - Role of the Component 11
        - Competitive Benchmarking 11
        - Why Cloudflare KV Was Chosen 12
        - Enhancement Opportunities 12
4. Data Flow Walkthrough 13
5. Design Principles 13
6. Tradeoffs 14
    - Serverless, Edge-First Architecture 14
    - Cloudflare D1 for Storage 14
    - Mock Data vs. Real Integrations 14
    - Client-Side Filtering and Aggregation 14
    - No Authentication 14

1. Architecture Overview

Architecture Summary

Cerebro is a product feedback dashboard that aggregates customer feedback from multiple channels and provides real-time triage, longitudinal analytics, and AI-powered insights to help product teams prioritize work and monitor product health

- It is built on a serverless JAMstack architecture
- Cloudflare Pages are used for global edge hosting of the React frontend
- Cloudflare Pages Functions are used for serverless API endpoints
- Cloudflare D1 provides structured, queryable storage for normalized feedback and derived analytics
- Cloudflare Workers serve as the core execution and API layer, handling feedback ingestion, querying, and aggregation

Assumptions

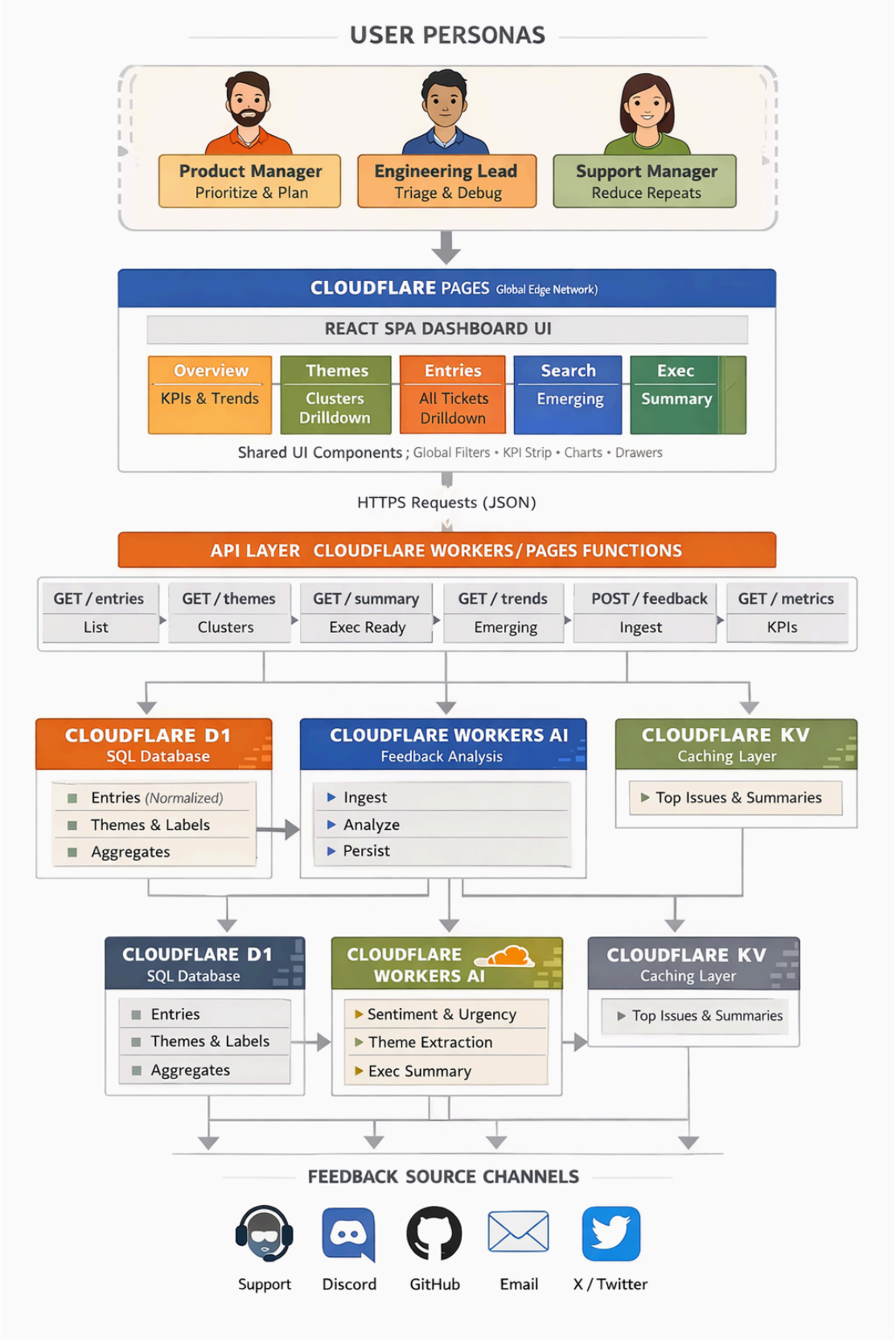
- Feedback & Ticket Data
    - No real ingestion pipelines, webhooks, or third-party API clients exist
    - No real-time updates. Only mock data is used
    - Sentiment, urgency, issue type, source, status, and customer segment come from fixed enumerations
- Database (D1)
    - Feedback is stored as JSON payloads in a simple relational table structure
    - D1 read replicas may exhibit slight lag; eventual consistency is acceptable
- Usage Patterns
    - Desktop-first UX
    - Users analyze recent data far more frequently than historical data
    - Interaction is read-only; users consume insights rather than edit data
- Business Logic
    - Time-based comparisons use symmetric windows
    - No authentication or authorization is needed at MVP stage

Core Persona

1. **Product Manager:** Owns product quality and prioritization by turning fragmented, multi-source user feedback into clear themes, trends, and roadmap decisions
2. **Engineering Lead:** Assesses the severity and impact of user-reported issues by drilling from high-level themes into concrete, reproducible feedback examples
3. **Customer Support Manager:** Identifies recurring customer issues and source patterns to improve response strategies and escalate systemic problems efficiently

Context & responsibilities ● Receives feedback from multiple fragmented channels ● Must continuously triage issues, identify patterns, and distinguish signal from noise ● Responsible for prioritizing fixes, influencing roadmap decisions, and communicating insights to stakeholders ● Operates in time-constrained environments (weekly reviews, launch windows, incident response)	**Jobs to be Done** ● Quickly understand *what users are complaining about* and *why* ● Identify emerging or escalating issues before they become incidents ● Prioritize work based on urgency, impact, and sentiment, not anecdotal evidence ● Communicate concise, evidence-backed insights to engineering leads and leadership
Key pain points ● Feedback is noisy, unstructured, and spread across tools ● Hard to quantify urgency and trend changes over time ● Manual synthesis into weekly summaries is time-consuming and error-prone ● Difficulty tracing high-level insights back to raw customer evidence	**How Cerebro helps** ● Aggregates and normalizes feedback into a single, consistent view ● Automatically clusters feedback into product-relevant themes ● Surfaces sentiment, urgency, and emerging issues at a glance ● Enables drill-down from dashboards to individual tickets for explainability ● Produces executive-ready summaries for stakeholder communication

2. High-Level Architecture Diagram



3. Component Breakdown

3.1 Cloudflare Pages (Frontend Hosting)

Role of the Component

Cloudflare Pages hosts the Cerebro frontend application, which serves as the primary interaction layer for users. It delivers the React-based dashboard UI, handles routing, and integrates seamlessly with Cloudflare Workers / Pages Functions to serve API-driven views

Pages enables fast, globally distributed delivery of static assets while allowing dynamic data fetching through tightly coupled serverless functions

Competitive Benchmarking

Dimension	Cloudflare Pages	Vercel	Netlify
Core focus	Static frontend + edge functions	Frontend-first (Next.js)	Static + serverless
Edge integration	Native with Workers & D1	Limited (Edge Functions, region-scoped)	Limited (Functions not truly edge)
Backend proximity	Same platform as Workers, AI, DB	Separate backend infra	Separate backend infra
Deployment model	Git-based, zero-config	Git-based, framework-optimized	Git-based
Observability	Limited UI visibility	Strong deployment previews	Moderate
AI-native workflows	Strong (Workers AI integration)	External AI services	External AI services
Best suited for	Full-stack Cloudflare-native apps	Next.js-heavy frontend apps	Jamstack sites

Why Cloudflare Pages Was Chosen

Cloudflare Pages was selected because it **minimizes integration friction** between the frontend and the rest of the Cloudflare Developer Platform. In Cerebro:

- The frontend UI, backend APIs (Workers), AI inference (Workers AI), and database (D1) all run within the same ecosystem
- Pages Functions allow frontend routes to directly invoke backend logic without cross-platform networking or authentication complexity
- Global edge deployment ensures low-latency interactions for dashboard filtering, searching, and drill-downs

Enhancement Opportunities

1. **Functions Visibility Panel** - Add a dedicated Pages dashboard section showing:
    - Detected Functions
    - Enabled/disabled status
    - Last deployed SHA
    - Health check and recent invocation count

2. **Local Dev Parity Mode** - Provide a `wrangler pages dev --functions` mode:
    - Runs Pages + Functions locally
    - Supports hot reload for both frontend and backend logic
      This would significantly close the iteration-speed gap with Vercel

3. **AI-aware Frontend Tooling** - Native support for:
    - Previewing AI-powered API responses (e.g., summaries, themes)
    - Mocking Workers AI outputs in local/dev mode. This would be especially valuable for AI-first dashboards like Cerebro

---

3.2 Cloudflare Workers (Core Execution Layer)

Role of the Component

Cloudflare Workers serve as the **core execution and API layer** for Cerebro. They handle all backend logic, including feedback ingestion, filtering and querying entries, aggregating metrics, detecting emerging themes, and serving dashboard APIs consumed by the frontend

Competitive Benchmarking

Dimension	Cloudflare Workers	AWS Lambda	Vercel Edge Functions
Execution model	Edge-first, globally distributed	Region-based	Edge, but frontend-focused
Cold starts	Near-zero	Noticeable	Low

Language/runtime	JS/TS (Web-standard APIs)	Multi-language	JS/TS
Native DB + AI bindings	D1, KV, Workers AI	External services	Limited
Orchestration	Native (Workflows)	Step Functions (external)	Limited
Cost model	Request-based, predictable	Invocation + duration	Request-based
Best suited for	API-heavy, global platforms	Backend-heavy services	Frontend augmentation

Why Cloudflare Workers Were Chosen

Cloudflare Workers were chosen because they allow the Cerebro backend to run **at the edge**, tightly integrated with storage, AI, and orchestration primitives. Key reasons:

- **Low-latency APIs:** Filtering, search, and drill-down interactions feel instantaneous, even with large datasets
- **Platform-native integration:** Workers can directly bind to D1, Workers AI, Workflows, and KV without additional SDKs or credentials
- **Clear mental model:** A single execution layer handles both ingestion and read paths, reducing architectural complexity for a PM-built prototype
- **Production parity:** The same Workers code can scale from mock data to real integrations without architectural changes

Enhancement Opportunities

1. **First-class API Observability** - Add a Workers dashboard view showing:
    - Endpoint-level latency and error rates
    - Request volume by route
    - Slow query detection for D1-backed handlers

2. **Pre-deployment Validation** - Introduce a `wrangler workers validate` step:
    - Verifies bindings (D1, AI, KV, Workflows)
    - Flags missing environment variables
    - Catches common runtime misconfigurations early

3. **AI-aware API Debugging** - Allow Workers to expose:
    - Structured AI inference traces)
    - Cost and latency breakdown per AI call. This would significantly improve debuggability for AI-powered endpoints like summaries and theme extraction

---

3.3 Cloudflare D1 (Structured Storage)

Role of the Component

Cloudflare D1 serves as the **primary system of record** for Cerebro. It stores all structured data required to power the dashboard, including:

- Normalized feedback entries from multiple sources
- AI-generated metadata (sentiment, urgency, theme assignments)
- Derived aggregates used for KPIs, trends, and emerging theme detection

Competitive Benchmarking

Dimension	Cloudflare D1	AWS DynamoDB	Supabase
Data model	Relational (SQL)	Key–value / NoSQL	Relational (Postgres)
Query flexibility	High (joins, filters, aggregates)	Limited	High
Serverless integration	Native with Workers	Requires SDK & IAM	External service
Operational overhead	Very low	Medium	Medium
Analytics suitability	Strong for MVP / dashboards	Weak (needs ETL)	Strong but heavier
Latency profile	Edge-adjacent	Region-based	Region-based
Best suited for	Structured, query-heavy apps	High-scale KV workloads	Full backend apps

Why Cloudflare D1 Was Chosen

D1 was chosen because Cerebro's core data is **structured, relational, and analytics-heavy**, not object-based or event-stream oriented. Key reasons:

- **SQL-first querying:** Enables flexible filters and aggregations without additional processing layers
- **Tight Worker integration:** Workers can query D1 directly without networking overhead, credentials, or external drivers

- **Low operational burden:** No database provisioning, scaling, or connection management - ideal for rapid prototyping
- **Explainability:** Relational schemas make it easy to trace dashboard insights back to raw feedback entries

Enhancement Opportunities

1. **Query & Limit Transparency** - Add tooling to surface:
    - Per-query execution time
    - Statement and batch limits
    - Throttling reasons with actionable error messages
2. **Data Quality & Freshness Dashboard** - Provide a D1 console view showing:
    - Record counts by table
    - Timestamp ranges (oldest/newest records)
    - Schema version and migration status
3. **Analytics-Friendly Bulk APIs** - Introduce bulk insert/update helpers optimized for:
    - Seeding datasets
    - Periodic aggregation jobs
    - Analytics-heavy ingestion patterns

3.4 Cloudflare Workers AI (AI Insights)

Role of the Component

Cloudflare Workers AI powers **Cerebro's AI-driven insights layer** for:

- Analyze feedback data to identify **emerging themes**, trends, and anomalies
- Generate **summarized insights** across sentiment, urgency, issue types, and customer segments
- Produce **actionable recommendations** surfaced in the AI Insights overlay
- Augment rule-based analytics with **context-aware natural language interpretation**

Competitive Benchmarking

Dimension	Cloudflare Workers AI	OpenAI API	AWS Bedrock	Vertex AI
Deployment model	Edge-native, serverless	External API	Managed cloud service	Managed cloud service

Integration with compute	Native with Workers	Requires outbound HTTP	Requires IAM + SDK	Requires IAM + SDK
Latency profile	Very low (edge-executed)	Internet-bound	Region-bound	Region-bound
Operational overhead	Very low	Low	Medium	Medium
Cost transparency	Limited today	High	Medium	Medium
Model access	Curated CF models	Broad model choice	Curated	Broad
Best suited for	Inline product intelligence	General LLM workloads	Enterprise AI pipelines	ML-heavy platforms

Why Cloudflare Workers AI Was Chosen

Workers AI was selected because Cerebro's AI use case prioritizes **proximity, explainability, and operational simplicity** over raw model flexibility. Key reasons:

- **Edge-adjacent execution**: AI inference runs close to where data is processed, minimizing latency and avoiding external network calls
- **Zero credential management**: No API keys, secrets, or IAM roles required. AI is bound directly to the Worker runtime
- **Tight architectural fit**: Workers AI integrates naturally into the existing Pages + Functions + D1 stack

Enhancement Opportunities

1. **Binding & Configuration Transparency**
    - Provide CLI visibility into AI bindings and status (`wrangler pages ai status`)
    - Expose AI enablement and model configuration in deployment logs
2. **Observability & Debuggability**
    - Add AI usage metadata to API responses (AI vs fallback, model used)
    - Expose AI success/failure indicators in the dashboard
3. **Local Development & Prompt Iteration**
    - Support Workers AI in `wrangler pages dev` with mocks or emulators
    - Enable local prompt testing and response inspection

3.5 Cloudflare K,V Cache (Model response and data refresh)

Role of the Component

Cloudflare KV is used in Cerebro as a **selective caching and performance optimization layer** to support fast, repeat access to derived and aggregated data that powers the dashboard experience. Based on the user guide workflows, KV is used to:

- Cache **derived aggregates** used across multiple views, such as:
  - KPI values (Total Tickets, Critical %, Top Source)
  - Pre-computed trend buckets for common time ranges (e.g., Last 7 days, Last 30 days)
- Reduce repeated recomputation for:
  - Overview page initial load
  - Business Metrics page comparisons
  - Needs Attention overlays (alerts & emerging issues)
- Improve responsiveness for **refresh-heavy workflows** (manual refresh button, frequent filter toggling)

KV is explicitly **not** used as a system of record. All authoritative data continues to live in D1, with KV acting as a best-effort acceleration layer for read-heavy dashboard paths

Competitive Benchmarking

Dimension	**Cloudflare KV**	**Redis (Managed)**	**DynamoDB DAX**
Data model	Key–value	Key–value	Key–value
Consistency	Eventual	Strong	Strong
Deployment model	Global edge replication	Region-based	Region-based
Integration with compute	Native with Workers	External service	External AWS service
Latency profile	Very low (edge reads)	Low (regional)	Low (regional)

Operational overhead	Very low	Medium	Medium
TTL control	Limited (min TTL enforced)	Fully configurable	Fully configurable
Best suited for	Edge caching, derived data	Stateful caching	High-scale AWS apps

Why Cloudflare KV Was Chosen

KV was chosen because Cerebro's access patterns are **read-heavy, repetitive, and tolerance-based** rather than strongly consistent

Key reasons:

- **Edge-adjacent performance:** KPI strips, trend charts, and overlays are accessed repeatedly during triage workflows; KV allows these to load instantly without re-querying D1
- **Native Workers integration:** KV bindings are accessed directly inside Pages Functions with no SDKs, credentials, or network hops
- **Clear separation of concerns:**
  - D1 → source of truth for tickets and metadata
  - KV → cache for aggregates, summaries, and derived outputs
- **Manual refresh model compatibility:** Since users explicitly refresh data (🔄 button), KV staleness is acceptable within short TTL windows
- **Low operational complexity:** No cache cluster management or scaling decisions required

KV aligns well with Cerebro's **dashboard-first UX**, where speed and responsiveness matter more than immediate consistency

Enhancement Opportunities

- **Cache Transparency & Debuggability** - KV currently behaves as a black box
  - Expose cache status in the UI (e.g. "Serving cached data - updated 2m ago")
  - Distinguish clearly between:
    - Cache miss
    - Empty cache
    - Cache error

- **Observability & Performance Metrics**
    - Surface cache hit/miss ratios for key dashboard endpoints
    - Add warnings for uninitialized or misconfigured namespaces

---

4. Data Flow Walkthrough

- **Ingestion -** Feedback from multiple sources is submitted (via mock inputs or manual entry) to an ingest API hosted on **Cloudflare Workers**, where it is normalized and stored in **Cloudflare D1**
- **Orchestration** - A **Cloudflare Workflow** is triggered asynchronously to process the new feedback entry, decoupling ingestion from analysis and ensuring reliable multi-step execution
- **AI Analysis** - The Workflow invokes **Workers AI** to classify sentiment and urgency and to assign or update feedback themes. AI outputs are returned as structured data.
- **Dashboard APIs** - The frontend queries **Workers APIs** (e.g. themes, entries, trends, summaries), which read from D1 (and KV where applicable) and return filtered, dashboard-ready responses
- **Presentation & Drill-Down** - The dashboard, hosted on **Cloudflare Pages**, renders KPIs, charts, and tables, enabling users to drill down from high-level insights to individual feedback entries with full traceability

5. Design Principles

Cloudflare-native by default	Every architectural component maps directly to a Cloudflare product capability, minimizing external dependencies and maximizing platform cohesion
Separation of concerns	Ingestion, analysis, storage, and presentation are clearly decoupled, enabling reliable pipelines, simpler debugging, and faster iteration
Explainability over black-box AI	All AI-generated insights are traceable back to underlying feedback entries, ensuring transparency, trust, and actionable decision-making

6. Tradeoffs

Serverless, Edge-First Architecture

- **Decision:** Serverless was chosen to maximize development speed and operational simplicity; performance and limits are acceptable for dashboard-style workloads
- **Tradeoff:**
    - Zero infrastructure management vs. execution constraints (e.g. Pages Functions time limits
    - Automatic scaling vs. potential cold starts (largely mitigated by Cloudflare's global edge

Cloudflare D1 for Storage

- **Decision:** D1 was selected for its SQL familiarity, low operational overhead, and suitability for structured, query-heavy analytics.
- **Tradeoff:**
    - SQL-based, edge-distributed storage vs. eventual consistency across replicas
    - Simplicity and tight platform integration vs. portability outside the Cloudflare ecosystem

Mock Data vs. Real Integrations

- **Decision:** Mock data enabled faster iteration and clearer evaluation of product insights; real integrations are deferred to a production phase
- **Tradeoff:**
    - Rapid prototyping vs. production-grade ingestion complexity
    - Controlled test scenarios vs. real-world data variability

Client-Side Filtering and Aggregation

- **Decision:** Client-side filtering improves UX for the current scale (<10K entries). Server-side aggregation can be introduced as data volume grows
- **Tradeoff:**
    - Instant UI responsiveness vs. loading a bounded dataset into the browser
    - Fewer server round-trips vs. increased client memory usage

No Authentication

- **Decision:** Authentication was intentionally deferred to focus on core feedback intelligence. It is required for any production deployment
- **Tradeoff:** Faster setup vs. lack of access control and security

---