Round 1 - AI-Native Solution Crafting Test

Company	Cyepro Solutions
Problem	Notification Prioritization Engine
Dead Line	25th FEB, 6:00 PM

Problem Statement

You are designing a **Notification Prioritization Engine** for a product that sends notifications to users from multiple features (messages, reminders, updates, alerts, promotions, system events, etc.).

Currently, users receive too many notifications. Some are repetitive, some arrive at bad times, and some low-value notifications are sent while important ones are missed or delayed.

Design a system that decides, for each incoming notification event, whether it should be sent **Now**, **Later** (deferred/scheduled), or **Never** (suppressed).

Core Requirements

- Classify each notification event into one of: Now / Later / Never.
- Prevent duplicate notifications (exact duplicates and near-duplicates).
- Reduce alert fatigue by considering recent notification history and frequency.
- Handle conflicting priorities (for example, urgent but potentially noisy notifications).
- Support human-configurable rules without requiring a full code deployment for every change.
- Log a clear explanation for each decision (why sent now, delayed, or suppressed).
- Fail safely if AI or a dependent service is slow/unavailable.

Input Shape

Each incoming notification event may include fields such as:

- user_id
- event_type
- message or title
- source or service
- priority_hint (optional)
- timestamp
- channel (push / email / SMS / in-app)
- metadata (context-specific fields)
- dedupe_key (optional; may be missing or unreliable)
- expires_at (optional)

Constraints

- High event volume (for example, thousands per minute).
- Low-latency decisions are preferred for most requests.
- Some notifications are time-sensitive and become useless if delayed.
- Some notifications are optional/promotional and should be deprioritized during noisy periods.
- The same user may receive events from multiple services at nearly the same time.
- Duplicate keys may be missing or unreliable.
- The system must be explainable and auditable.
- Important notifications should not be silently lost.

Candidate Deliverables

- High-level architecture (components/services + data flow)
- Decision logic design (Now / Later / Never classification strategy)
- Minimal data model (events, history/counters, suppression/defer records, audit logs)
- API or service interfaces (3 - 5 endpoints/contracts)
- Duplicate prevention approach (exact and near-duplicate handling)
- Alert fatigue strategy (cooldowns, caps, batching, digest, or other design)
- Fallback strategy when AI/model/service is slow or unavailable
- Metrics and monitoring plan

What We Evaluate

- Solution thinking and practical tradeoff decisions
- Clarity and completeness of design
- Edge-case handling and reliability mindset
- Explainability and auditability
- Communication quality (structured, concise, logical)

How to Submit

- Push your Solution into your GitHub.
- Make sure your repo is public.
- Record a walk-through video of your solution.
- Share you submissions to varun@cyepro.com, hr-admin@cyepro.com

Allowed Tools

Internet and AI tools (Claude Code / ChatGPT / Copilot / etc.) are allowed. If used, candidates should briefly mention what they used and what they changed manually.