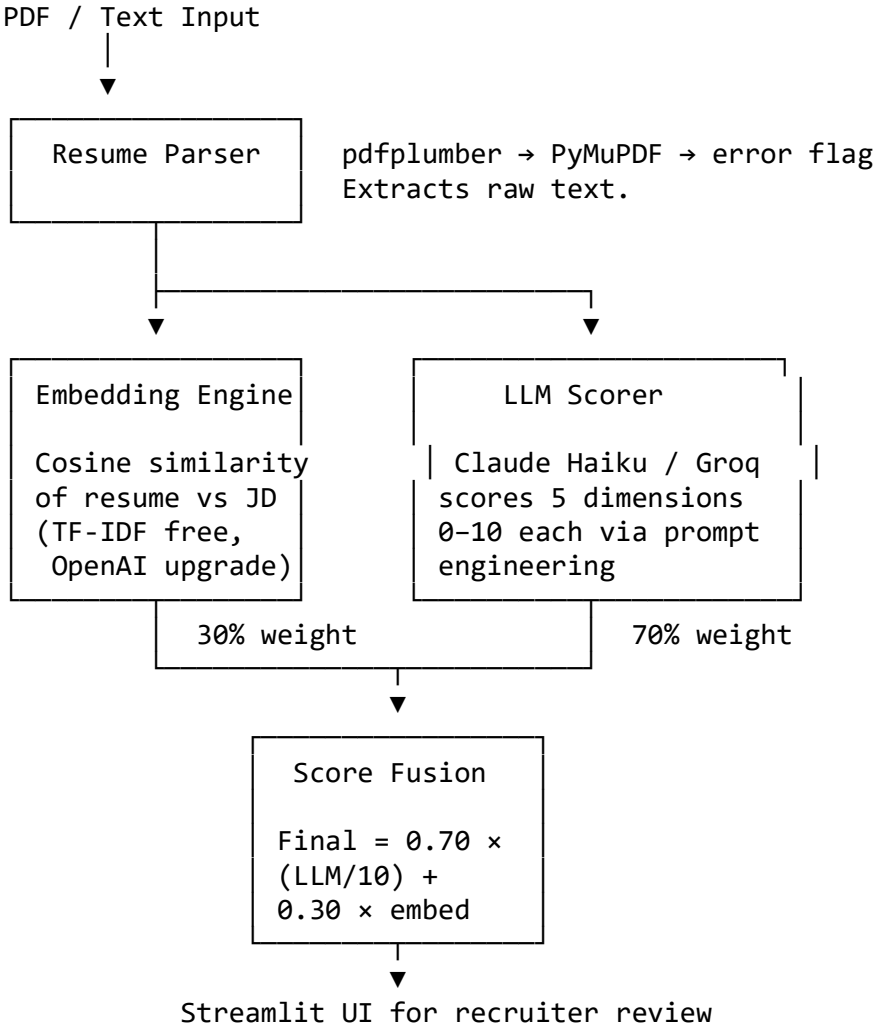
Resume Screening System — Design Document

The Problem, Restated

200+ resumes. No HR team. 30 days to ship a product. The constraint is not "how do we read resumes faster" but efficiently we can read resumes. So, I built the solution the same way I'd build your product.

Architecture



Why Two Signals, Not One

Most naive approaches use either keyword matching (fast, gameable) or LLM scoring alone (slow, expensive). I combined both deliberately:

Signal 1 - LLM Rubric (70%) The LLM reads the full resume and scores five dimensions that map directly to what AI Know A Guy said they value: technical depth, AI/ML hands-on experience, shipping velocity, systems thinking, and communication clarity. Temperature is set to 0 for consistent, deterministic output. The prompt anchors scores strictly to resume evidence not to inferred assumptions to reduce hallucination risk.

Signal 2 - Semantic Similarity (30%) Cosine similarity between the resume and job description. Catches resume that are topically aligned but may have used different phrasing. This also serves as a

zero-cost fallback when no API key is available the system still ranks candidates meaningfully using TF-IDF, a fully self-contained implementation requiring no external dependencies.

Why 70/30? The LLM reads intent and evidence. The embedding reads topic overlap. A candidate can keyword-stuff their resume to fool an embedding but can't fool a model asked, "show me the metrics." Conversely, a strong candidate who uses different vocabulary than the JD deserves a boost hence the 30% semantic floor.

The Rubric — Design Choices

Dimension	Weight	Reasoning
Technical Skills	30%	Non-negotiable foundation for an engineering role
AI/ML Depth	25%	The product is AI surface-level exposure is not enough
Shipping Velocity	20%	"30 days to ship" is in the first line of the JD
Systems Thinking	15%	Differentiates senior from mid-level
Clarity	10%	Tie-breaker; also a proxy for communication on the job

The rubric is not generic. It was calibrated to the exact language of this job posting. A different JD would produce a different rubric. This is intentional **the system is a template, not a fixed scorer**.

Tradeoffs I Made Consciously

Cost vs. Quality Claude Opus would score more accurately. Claude Haiku costs ~$0.02 for 200 resumes and is accurate enough for a first-pass filter.

Speed vs. Accuracy on Embeddings TF-IDF is free and instant. OpenAI `text-embedding-3-small` is better but costs ~$0.001 for the batch. I built both and made the upgrade path one environment variable (`OPENAI_API_KEY`). The system chooses automatically.

Seniority vs. Output I chose not to penalise candidates who lack a "Senior" title. Instead, `shipping_velocity` and `ai_ml_depth` reward demonstrated output. A 3-year builder with deployed AI products scores higher than a 10-year title-holder with vague bullets. This aligns with AI Know A Guy's stated values.

What I didn't build (and why) A fine-tuned classifier would be more accurate but requires labelled data I don't have. An agent that scrapes GitHub or LinkedIn would add signal but raises privacy concerns and latency. Both are future improvements, not day-one requirements.

Known Failure Modes

Risk	What Happens	Mitigation
LLM hallucination	Model infers skills not in resume	`temperature=0`, prompt says "evidence only"
PDF layout breakage	Multi-column PDFs lose text structure	Paste-text fallback in UI; two parser libraries tried
Keyword stuffing	High embed score, low LLM depth	LLM asks for metrics not keywords