TECHNICAL PROPOSAL · PORTSWIGGER

Contract similarity & risk orchestration.

An AI-assisted triage layer that treats commercial contracts like version-controlled code — extracts them, compares them to precedent, and routes only the unusual or risky items to human review. The goal is to remove most of the manual load from boilerplate-heavy work while keeping legal in control of exceptions and high-risk cases.

SUBMITTED BY	AUDIENCE	STATUS	SCOPE
Tech submission	Legal · Eng · Security	Draft proposal	Standard commercial paper

01 / Problem

Most legal work isn't novel work.

Small in-house legal teams spend most of their time reading the same kinds of contracts — mutual NDAs, small order forms, standard commercial paper. Most of this material is repetitive, but each document still requires a full human read.

The useful question

Not *"can AI replace legal judgment?"* — but *"can AI reliably identify what is already standard, what is genuinely different, and what needs attention now?"*

That framing changes the whole design. We're not building a drafter or a redline assistant. We're building a triage layer.

Where the queue pressure comes from

- **End of month / end of quarter** — predictable spikes the team can't staff for.
- **Repeat counterparties** sending near-identical paper that still gets re-reviewed end-to-end.
- **Vendor-side redlines** that touch one clause but trigger a full re-read.
- **Boilerplate fatigue** — senior reviewers spend judgment on low-stakes work.

02 / Core idea

Baseline-to-delta.

Treat each contract as a delta from known-good precedent. If the delta is small, accelerate it. If the delta is meaningful, route it. If the delta is novel, escalate it.

- **Extract** the document into clean Markdown — diff-able, version-controllable.
- **Compare** clause-by-clause to a library of accepted boilerplate and historical exceptions.
- **Compare again** to a library of rejected or problematic language.
- **Review** ambiguous cases with multiple LLMs running independently.
- **Compare outputs** — agreement increases confidence, disagreement is an escalation signal.
- **Route** only unusual, high-risk, or conflicting cases to a human.

If it looks like something already accepted, it should move quickly. **If it deviates too much**, it should be flagged. The model never decides hard-fail criteria — those stay with legal, as deterministic rules.

03 / Pipeline

Six stages, end to end.

Each stage is replaceable in isolation. The system is a chain of small, auditable components, not a monolith.

01
Extract
PDF / DOCX → Markdown. Preserves clause hierarchy, dates, tables.

02
Similarity
Clause-level comparison against known-good and known-bad libraries.

03
Ensemble
Three LLM personas — Stickler, Commercial, Adversary — review in parallel.

04
Policy
Deterministic CI checks for forbidden phrases, jurisdictions, liability triggers.

05
Failure loop
Failed clauses compared to historical rejections and accepted exceptions.

06
Feedback
Knowledge-graph sync, plus 5% stochastic QA on auto-approvals.

04 / Operational logic

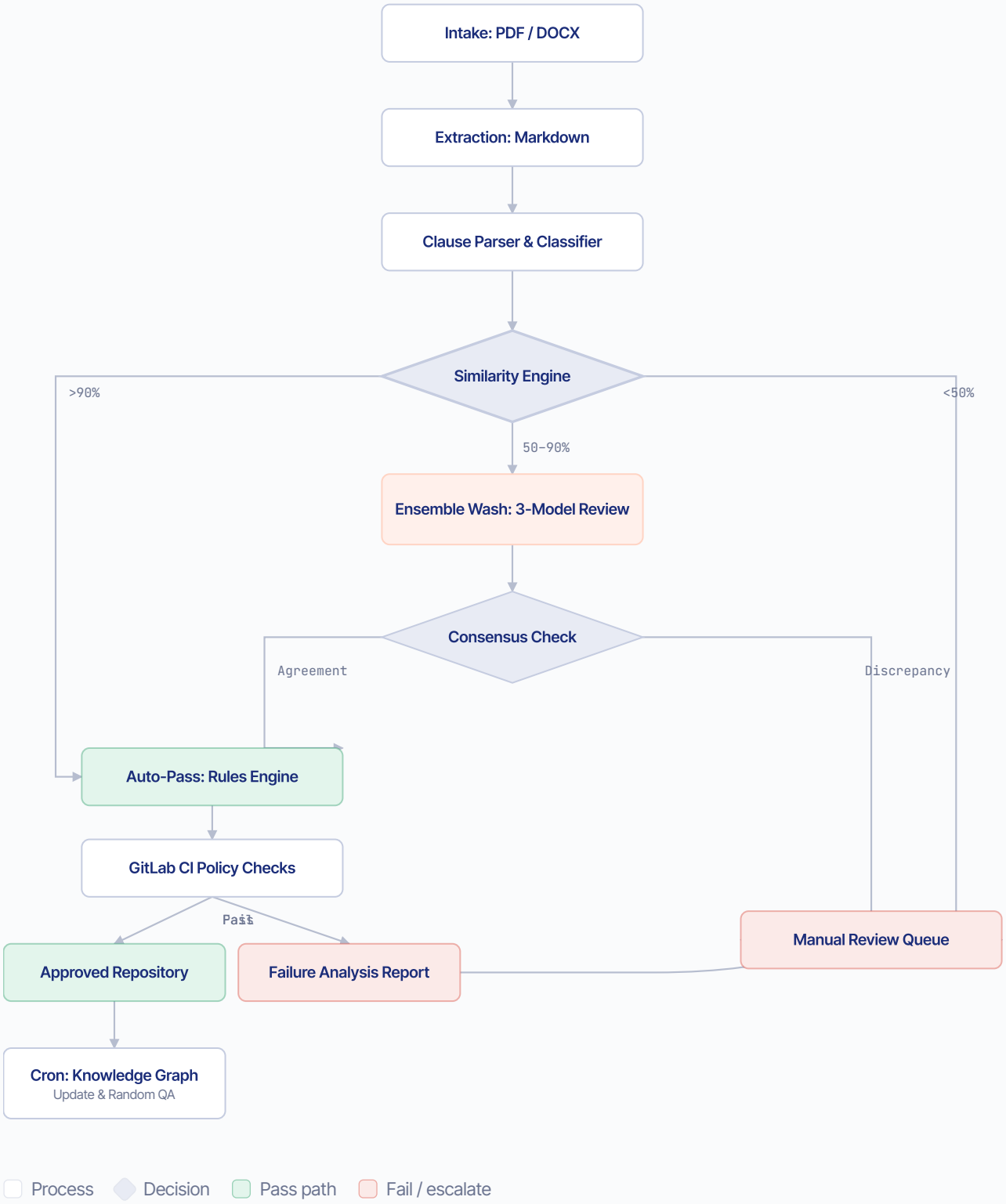
Similarity bands & routing.

The single most important decision the system makes is which band a document lands in. Everything else cascades from that.

SIMILARITY	STATUS	ACTION
90% - 100%	Standard	Bypass LLM review entirely. Route directly to deterministic rules engine.
50% - 89%	Hybrid	Route to 3-model ensemble. Consensus → rules; discrepancy → human.
Below 50%	Novel	Flag as high complexity. Route directly to manual review queue.

The system on one screen.

Click any node for an explanation of what it does and why it sits where it does.



ROUTING DECISION

Similarity Engine

Per-clause and overall similarity vs the precedent library. Three bands: >90% Standard → auto-pass path; 50–90% Hybrid → ensemble; <50% Novel → straight to human.

A consistent way to order the queue.

Per-clause deviation × commercial value × clause criticality. The formula doesn't need to be precise — it needs to order risk consistently and explainably.

$$R = \Sigma(D_i \times V \times C_i)$$

D_i per-clause deviation (0–1) **V** commercial value band (1–10) **C_i** clause criticality (0–1)

What it gives us

- Composable scoring — clause-level rolls into doc-level rolls into stakeholder-level.
- Cheap to compute. No LLM in the loop.
- Explainable — a lawyer can see which clause contributed which fraction.
- A small deviation in a high-value, high-criticality clause beats a big deviation in a low-stakes one.

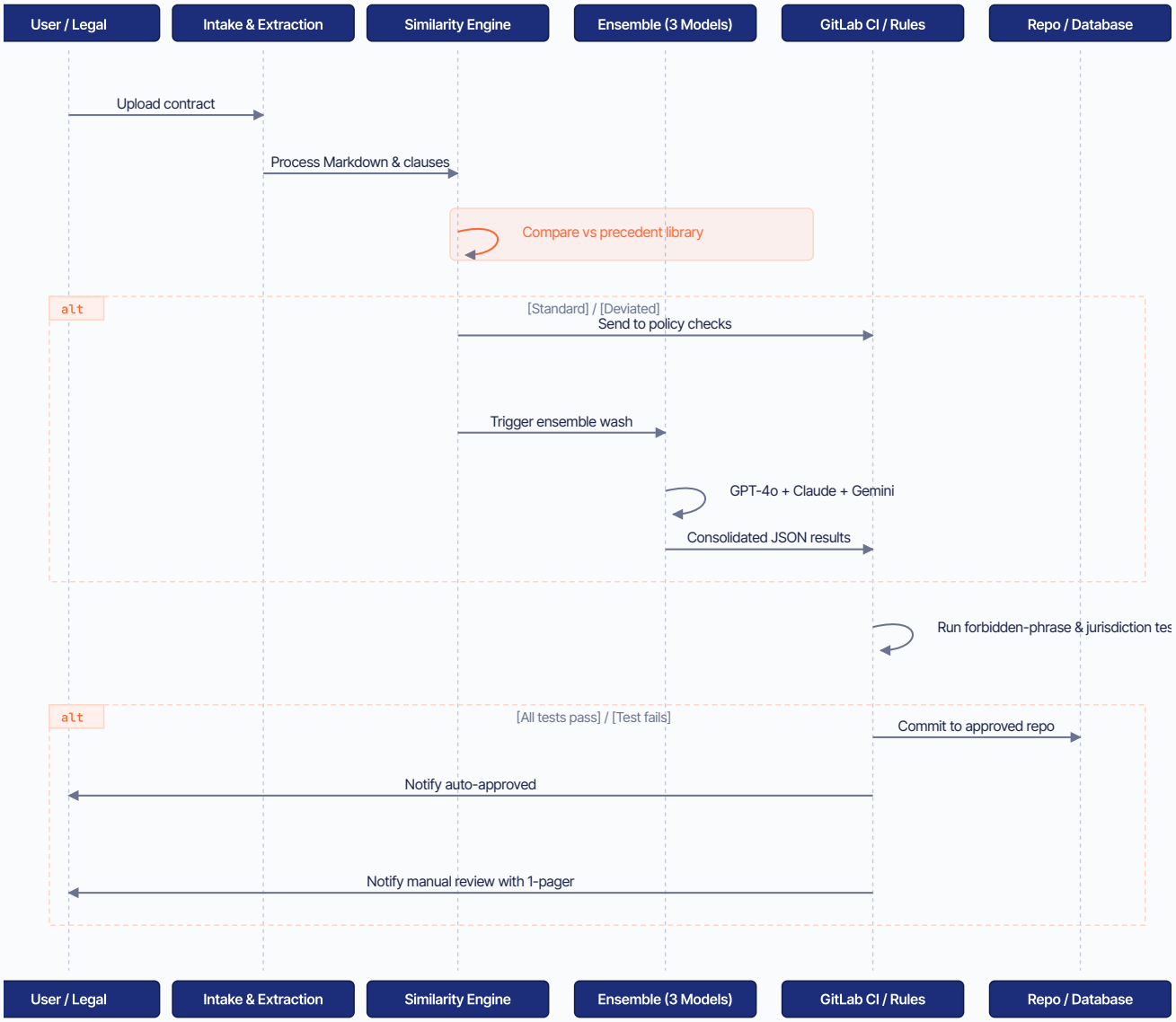
What it doesn't capture

- **D is similarity, not legal materiality** — closely correlated, not identical.
- Multiplicative interactions across clauses (weak cap + wide indemnity is worse than the sum).
- Stakeholder context — same clause from different counterparties means different things.
- Temporal risk — auto-renewal in year 3 isn't visible at signing.

Calibration approach. Replay the last ~100 human decisions and fit thresholds so the auto-pass band contains zero historical escalations. The formula gives us ordering; the data gives us cut-offs.

A single contract, lane by lane.

The two `alt` frames are the only branching logic — everything else is sequential.



STEP 3

Compare vs precedent library

Per-clause embedding similarity against the accepted + rejected precedent corpus.

08 / Ensemble review

Three reviewers. One input. Designed to argue.

Different models, different system prompts. Agreement is cheap — disagreement is the cheap escalation signal we want.

REVIEWER 01

The Stickler

Literal wording deviation.

- Missing standard clauses.
- Changed wording in indemnity, liability, termination.
- Clause reordering that signals non-standard drafting.
- Anything that doesn't match the known-good library.

→ JSON: similarity %, status per clause, risk flags

REVIEWER 02

The Commercial

Business exposure, not wording.

- Liability cap size.
- Payment terms & auto-renewal risk.
- Termination rights.
- Spend & value band — what's actually at stake.

→ JSON: commercial risk score, value band, impact per clause

REVIEWER 03

The Adversary

Loopholes, ambiguity, hidden risk.

- Vague liability or indemnity language.
- Hidden carve-outs.
- Jurisdiction or data-transfer risk.
- Anything that could become a dispute later.

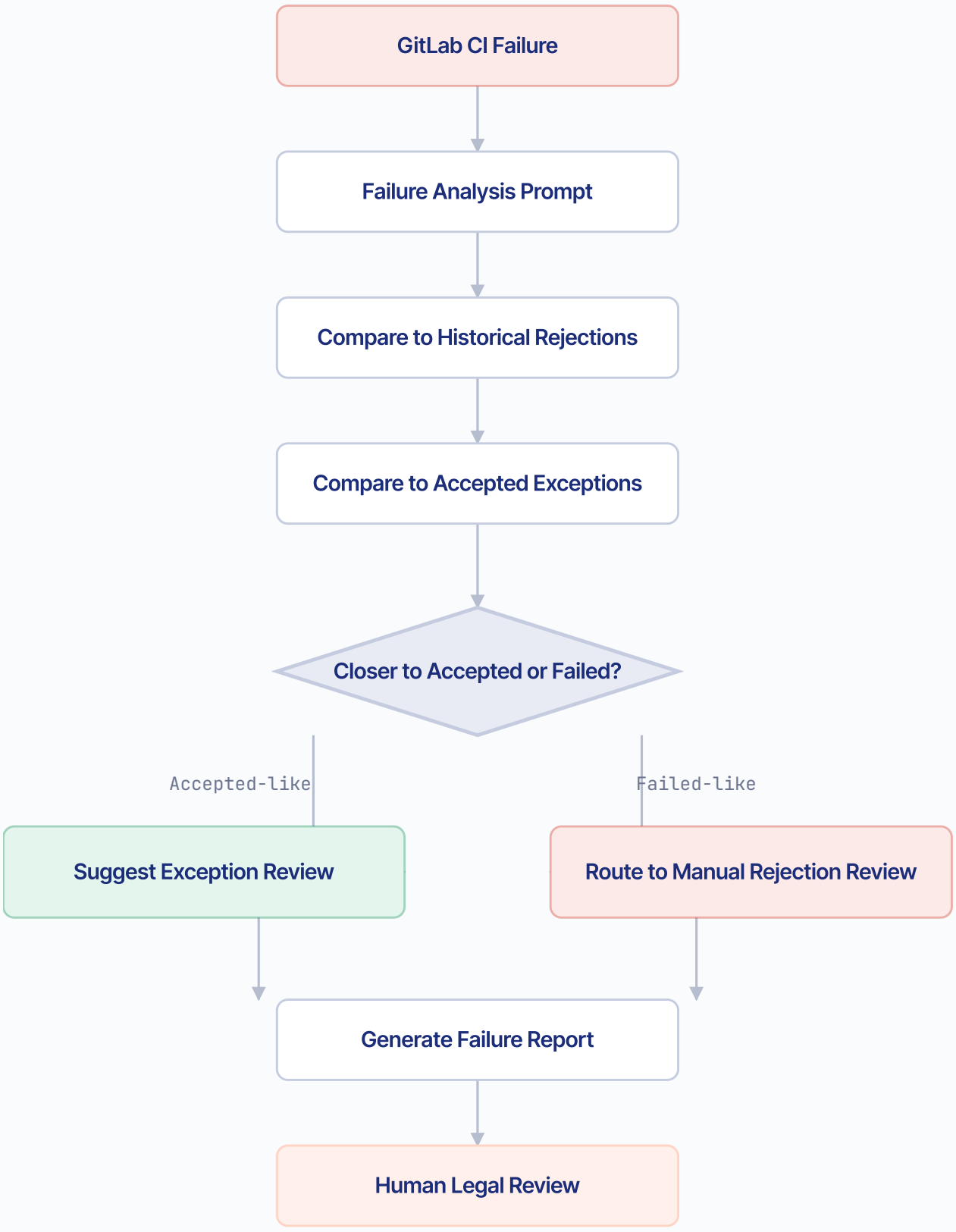
→ JSON: loophole score, issue type, severity

The orchestrator reads all three JSON outputs, finds 2/3 or 3/3 agreement, ranks disagreements by urgency, and decides: auto-pass, pass with warning, or manual review. Models output structured JSON only — no free text the system can't audit.

09 / Failure analysis

A failed check isn't a stop. It's a start.

Most policy failures aren't fatal — they're context-dependent. The loop tells you which way to lean before a human sees it.



DECISION

Closer to Accepted or Failed?

Distance to nearest rejected vs distance to nearest accepted exception. The shorter distance wins.

What we store, and why.

Three core entities and a knowledge graph linking them. Lean by design — every field exists to support an audit or a routing decision.

Document entity

FIELD	TYPE	PURPOSE
`document_id`	`string`	Unique identifier — primary key.
`document_type`	`string`	NDA, MSA, SOW, Order Form — used for bucketing.
`extracted_markdown`	`text`	Clean, diff-able representation of the source.
`similarity_score`	`float`	Overall match to accepted precedent.
`risk_score`	`float`	Weighted score, from the risk formula.
`value_band`	`integer`	1–10. Set by legal at intake.
`review_status`	`enum`	Pending · Approved · Failed · Escalated.

Clause entity

FIELD	TYPE	PURPOSE
`clause_id`	`string`	Unique identifier — clauses are first-class.
`clause_type`	`string`	Liability · indemnity · governing law · etc.
`similarity_score`	`float`	Per-clause match — the bit that actually matters.
`matched_good_id`	`string`	Closest accepted example, for context.
`matched_bad_id`	`string`	Closest rejected example, for context.
`deviation_flag`	`boolean`	Used by the rules engine for fast checks.

Model review entity

FIELD	TYPE	PURPOSE

FIELD	TYPE	PURPOSE
`review_id`	`string`	One per model run.
`model_name`	`string`	GPT-4o, Claude, Gemini.
`output_json`	`json`	Structured output — schema-validated.
`confidence`	`float`	Self-reported confidence, used as a signal not a gate.
`consensus_status`	`enum`	Agree · Disagree · Partial.

The **knowledge graph** sits on top of these and stores relationships: *"Stakeholder A accepted Clause B under Condition C"*, *"Clause X failed because of liability cap overreach"*, *"Project Z produced an exception that was later reused"*. The graph is what lets the system remember *not just what was seen, but what was accepted, rejected, or overridden*.

11 / Stochastic QA & governance

The 5% that keeps the system honest.

Auto-approval is the most dangerous part of the system. Stochastic QA is the cheapest defence against silent over-automation.

QA sampling

- **5% of auto-approvals** selected at random for blind human review on a cron.
- QA outcomes fed back into precedent memory.
- Sampling rate should vary by risk band — higher for high-value contracts.
- If the false-negative rate trends up, thresholds tighten automatically (with legal sign-off).

Security & governance

- Contract text treated as **untrusted input**. Never concatenated into system prompts.
- Source files & extracted Markdown encrypted at rest.
- Prompt versions and policy rules versioned & approved through PR review.
- Legal owns hard-fail criteria. Engineering owns the pipeline. Neither can override the other in production.

12 / First slice

What I would build first.

A two-week thin end-to-end slice. Not the system — just enough to prove the core flow works on real documents.

Ship in v1

- Markdown extraction (PyPDF / python-docx + formatter).
- Cosine similarity on a precedent set of ~10 NDAs.
- Single-model review with structured JSON output.
- Thin upload UI + CSV export of flags.
- One real legal decision made by week 2.

Defer to v2

- Three-model ensemble wash.
- Knowledge graph & relationship indexing.
- GitLab CI pipeline.
- Stakeholder profiles and learned exceptions.
- Dashboard & lawyer-facing triage UI.

The deliverable isn't the prototype. The deliverable is one auto-decision the legal team would otherwise have made manually, plus a 1-pager telling them what to build next.

Project Immunity

CONTRACT SIMILARITY & RISK ORCHESTRATION · V1.0 DRAFT

Submission for **PortSwigger**
Roving AI builder · 2026