Scope (from the diagram)

- **Your START:** when the **Market Analysis Agent** (fed by Search/Data-Collection) emits a **company_schema v1.0 JSON record**.
- **Your END:** a persisted **scoring payload** (score + breakdown + reason text + risk/feasibility flags), exposed via **REST** to:
    - **Opportunity Validation Agent** (gate/triage),
    - **Report Generation Agent** (the “why” panel),
    - **Results DB / Memory** (audit, cohort stats).

Not your scope: agent orchestration, crawling/scraping, NLP extraction, knowledge graph, UI.

Phase 0 — Contracts & Scaffolding (your entry handshake)

Goal: Make the inputs/outputs unambiguous so upstream and downstream can integrate without you.

Entrypoint: company_schema v1.0 (from Market Analysis Agent).


```
{
  "company_id": "lawfirm-123",
  "domain": "examplelaw.com",
  "digital": { "pagespeed": 78, "crm_flag": true, "ecom_flag": false },
  "ops": { "employees": 35, "locations": 2, "services_count": 6 },
  "info_flow": { "daily_docs_est": 240 },
  "market": { "competitor_density": 14, "industry_growth_pct": 6.1, "rivalry_index": 0.42 },
  "budget": { "revenue_est_usd": 4200000 },
  "meta": { "scrape_ts": "2025-08-07T17:15:00Z", "source_confidence": 0.83 }
}
```

Exit: validator + example fixtures everyone can test against.

What you build

- **company_schema v1.0** (strict types/ranges/null policy) and a JSON-Schema/Pydantic model.
- **weights.yaml v0** with the five weights + comments on meaning and change control.
- **Fixture set** (JSONL) with realistic and edge-case companies.
- **CI contract tests** that fail on schema drift.

Done when

- Upstream can run validate --in companies.jsonl and see clear errors/success.
- You've frozen company_schema@1.0.0 (minor bumps allowed, with tests).

Phase 1 — Normalization & Cohort Stats

Goal: Turn raw fields into **unitless, comparable** features deterministically.

Entrypoint: validated company records.

Exit: normalized feature vector + a **NormContext** (mean/σ snapshots).

What you build

- zscore, log10p, flag coercion, clipping, null policy (documented).
- **Cohort stats capture** (mean/std per field) for each batch; ID the stats used (e.g., norm_stats_id).
- Strict handling of missing/low-confidence inputs (warnings, never silent guesses).

Done when

- Re-running the same batch with the same NormContext yields identical features.

Phase 2 — Sub-scores (D/O/I/M/B)

Goal: Compute the five sub-scores exactly as spec + explanations.

Entrypoint: normalized features + flags.

Exit: {value ∈ [0,1], inputs_used, warnings} for each of D/O/I/M/B.

What you build

- **D (Digital Maturity):** 0.4*PageSpeed_scaled + 0.3*CRM + 0.3*Ecom.
- **O (Ops Complexity):** z(employees) + z(locations) + z(services).
- **I (Info Flow):** log10(daily_docs+1)/4.
- **M (Market Pressure):** z(comp_density)+z(industry_growth)-z(rivalry).
- **B (Budget):** log10(revenue)/7.
- Each returns value + what drove it (inputs/fallbacks).

Done when

- Golden tests (hand-calc'd) match within 1e-6 and every output has explain bits.

Phase 3 — Opportunity Score + Explainability

Goal: Produce the 0–100 score and a human-readable **why**.

Entrypoint: 5 sub-scores + weights.

Exit: scoring payload for a single record.

What you build

- **Final:** Score = 100*(0.25D + 0.20O + 0.20I + 0.20M + 0.15B) with consistent rounding.
- **Contributions:** points = weight*value*100, sorted; top drivers.
- **Reason text (templated, deterministic):** e.g., “High digital maturity (fast site, CRM) and strong document volume; budget moderate.”
- **Warnings passthrough** (e.g., missing revenue treated as 0).

Done when

- Random samples read correctly to a human and match the numbers shown.

Phase 4 — Risk & Feasibility Gates (lightweight validators)

Goal: Provide triage signals to the **Opportunity Validation Agent**.

Entrypoint: scoring payload + input confidences.

Exit: risk and feasibility blocks attached to the payload.

What you build

- **Risk:** data confidence score, missing-field penalties, scrape volatility.
- **Feasibility:** boolean gates for “deployable now” (docs present, CRM/e-com/booking present, budget above floor).
- Clear reasons when a company is flagged.

Done when

- Validation Agent can filter: {score >= X AND feasibility.ok == true AND risk.low}.

Phase 5 — Batch Runner & Persistence

Goal: Score cohorts reproducibly and persist audit-ready outputs.

Entrypoint: JSONL of companies.

Exit: JSONL of scores + DB rows + a stored NormContext.

What you build

- Batch CLI: score --in companies.jsonl --out scores.jsonl.
- **NormContext persistence** (stats snapshot + ID).
- **Results DB writes** (score, breakdown, reason, risk, feasibility, norm_stats_id).

Done when

- Same input + same NormContext ⇒ identical output file checksum and DB rows.

Phase 6 — Scorer Service (REST) + Integration Points

Goal: Let downstream pieces call you synchronously/asynchronously.

Entrypoint: HTTP requests from agents/UI/backends.

Exit: JSON responses + webhooks (optional) + metrics.

What you build

- **FastAPI** endpoints:
    - POST /score (single record), POST /score/batch (≤1k), GET /healthz, GET /stats (last NormContext).
- **Schema validation** with precise error messages.

- **OpenAPI** with copy-paste examples for Report Gen & Validation Agents.

Done when

- Contract tests pass; p99 <150ms/record locally; clean error semantics.

Phase 7 — Back-testing & Calibration (freeze the math)

Goal: Show the score tracks outcomes; tune weights if helpful.

Entrypoint: 200-SMB labeled cohort + your scores.

Exit: calibration report + frozen weights.yaml v1.0.

What you build

- Metrics: Spearman/Kendall rank corr, AUC (if binary), Precision@K/Lift.
- One-pager with plots and a go/no-go on weight tweaks.
- Tag the release (weights@1.0, scorer@1.0.0).

Done when

- Evidence is clear, weights are frozen, and results are reproducible.

Phase 8 — Sensitivity, Robustness & Drift

Goal: Prove stability and catch input drift early.

Entrypoint: baseline run.

Exit: stability report + alerts.

What you build

- **±10% weight sweeps** (sum preserved) → Kendall τ vs. baseline; top-K overlap.
- **Input drift checks** (mean/σ deltas, null-rate alarms).
- Thresholds that page you only when it actually matters.

Done when

- Stability documented; drift alarms tested; thresholds agreed with the lead.

Phase 9 — Observability & QA (operational hardening)

Goal: Make it safe to run without hand-holding.

Entrypoint: service + batch jobs.

Exit: actionable logs/metrics and failure modes that are obvious.

What you build

- Structured logs (latency, warning rates, nulls), histograms per field.
- Fuzz/mutation tests on formulas and schema.
- SLOs + light runbook (deploy, rollback, change control for weights/schema).

Done when

- A bad payload fails fast with a helpful message; on-call can triage from logs alone.

Phase 10 — Downstream Enablement (your exit handshake)

Goal: Make your output drop-in for the **Opportunity Validation** and **Report Generation** Agents.

Entrypoint: your REST and DB rows.

Exit: tiny SDKs/docs; no extra logic needed elsewhere.

What you build

- Mini-SDK helpers (score_record, score_batch) + curl examples.
- Output examples the Report Gen Agent can render as-is (score, breakdown, contributions, reason, risk/feasibility).
- “Hello-world” doc to score a JSONL in <15 minutes.

Done when

- Both agents call you in their pipelines without requiring changes to your code.

Quick ownership map (one line each)

- **In from:** Market Analysis Agent → company_schema v1.0.
- **You:** Normalize → Sub-scores → Score (0–100) → Explain → Risk/Feasibility → Persist → Serve.
- **Out to:** Opportunity Validation Agent, Report Generation Agent, Results DB/Memory.

Want me to spit out the OpenAPI for POST /score and a company_schema v1.0 draft next so the agent team can wire against it?