Architecture: Pluggable AI Platform for Job Board + Internet-wide Job Search

Below is a clear, implementable architecture that separates **core platform logic** from **pluggable AI providers** (paid/free). It supports:

- an **internal AI** for parsing, chat, and candidate matching (you control training/versioning), and
- a **Provider Manager** that routes/rotates external LLMs or search-enrichment models (OpenRouter, ChatGPT, Together, etc.) for internet search, enrichment, summarization, fallback and cost control.

I'll cover: high-level components, data flow, integration interfaces, model lifecycle & training, provider management, security/ops, and admin features. No code — just clear engineering specs you can hand to devs.

1. High-level Components (overview)

```
flowchart LR
  Client[Web / Bot Clients] --> Orchestrator[Chat Orchestrator / API Gateway]
  Orchestrator -->|search request| Aggregator[Meta-Search & Aggregation Engine]
  Aggregator --> ProviderMgr[Provider Manager (external LLMs & SERP)]
  ProviderMgr --> ExternalProviders[(OpenRouter / ChatGPT / Together / Search APIs)]
  Aggregator --> Cache[Search Cache/Index]
  Aggregator --> Normalizer[Job Normalizer & Dedupe]
  Normalizer --> JobsUnifiedDB[(jobs_unified DB)]
  Orchestrator --> InternalAI[Internal Model Service (Parsing / Chat / Matching)]
  InternalAI --> TrainingPipeline[Model Training / Fine-tune Pipeline]
  Orchestrator --> AppDB[(Application DB: jobs, candidates, applications)]
  AppDB --> Storage[Resume Storage (S3/Blob)]
  AdminConsole[Admin UI] --> ProviderMgr
  AdminConsole --> Metrics[Observability & Billing]
```

2. Component Responsibilities

2.1 Orchestrator / API Gateway

- Single entry point for web & bot clients.
- Handles auth, rate limits, session state, routing to appropriate subsystems (search, apply, profile).
- Exposes REST/GraphQL endpoints for UI and bot flows.

2.2 Meta-Search & Aggregation Engine

- Accepts candidate query (role, skills, location, profile).
- **Query Planner**: builds source-specific queries.
- Calls **Source Adapters** in parallel:
    - Your job board DB
    - Aggregator APIs (Jooble/Jora/Adzuna)
    - Company ATS APIs / career pages (Greenhouse/Lever/Workday)
    - SERP / Programmable Search (Google) where allowed
- Receives raw responses, sends to Provider Manager for LLM-based extraction/enrichment where needed, or uses internal parser.
- Dedupes, normalizes into `JobUnified` shape, ranks, caches results.

2.3 Provider Manager (PLUGGABLE)

- **Central piece** for external AI providers. Responsibilities:
    - Register providers: name, type (LLM/SERP), endpoint, credentials, model names, cost metrics, per-provider limits.
    - **Adapter interface** so each provider can be plugged/unplugged via config.
    - Routing & rotation policies (weighted round-robin, failover on error/latency/quality).
    - Per-call metrics: latency, tokens, cost estimate, success rate.
    - Hard limits & soft quotas (per-tenant / per-org).
    - Fallback chain: on failure / quality threshold, call next provider.
    - Expose audit metadata to UI (`provider`, `model`, `fallback_from`, `attempts`, `latency`).

Why pluggable? you can change providers, balance cost/quality, and add paid offers without rewriting core logic.

2.4 Internal AI Service (Owned)

- **Core functions**:
    - Resume parsing into structured fields (work history, education, skills).

    - Candidate-candidate matching & scoring.
    - Conversational assistant (Sree bot) for profile & apply flows.
    - Policy: handles PII-sensitive tasks inside your infra (you control training and data).
- **Model lifecycle**: versioned models, can roll back, A/B test.
- Connects to Training Pipeline for incremental fine-tuning using labeled data (your curated resumes + parser corrections).

2.5 Normalizer & Dedupe

- Converts provider output into canonical job object:
    - `job_id`, `title`, `company`, `locations[]`, `description`, `salary`, `posted_at`, `source_name`, `source_url`, `apply_url`, `origin_type`
- Dedupe using URL canonicalization + fuzzy textual similarity.
- Tag alternate sources for same job (displayed as “Also found on: X, Y”).

2.6 Databases & Storage

- **AppDB (Relational)** — jobs (your board), applications, candidates, recruiters, activity logs.
- **JobsUnifiedDB (NoSQL/Elastic)** — normalized internet-wide jobs for fast search.
- **Object Storage** — resumes, extracted files; kept secure and linked to AppDB.
- **Cache** — Redis for query-level caching, per-query TTL.
- **Metrics & Billing DB** — store provider usage, token/call costs.

2.7 Admin Console

- Provider management (add/edit/delete), set weights & limits.
- Observe provider metrics, fallback rates, search quality, cached hits.
- Manual override: disable provider, adjust weights.

2.8 Training Pipeline & Data Ops

- Data labeling store (human corrections of parse results).
- Fine-tuning orchestrator (automates dataset build, training, validation, bake into model version).
- Model registry: store versions, metadata (sha, date, dataset), promote to production, can rollback.

3. Key Data Shapes (names only)

- `JobUnified`: job_id, title, company, locations[], description_short, skills[], salary_min, salary_max, posted_at, source_name, source_url, apply_url, origin_type, dedupe_group_id, rank_score, provider_meta
- `Application`: application_id, job_id, candidate_id, answers{custom_q_id: answer}, resume_path, submitted_at
- `CandidateProfile`: candidate_id, name, email, phone, parsed_resume, skills[], work_history[], preferences, activity_log
- `ProviderConfig`: provider_id, name, type, endpoint, auth, weight, limits

4. Provider Adapter Interface (spec for devs)

Each provider plugin must implement the same logical interface:

1. `prepare_request(task, payload)` — convert internal task to provider-specific call.
    - `task` examples: `extract_fields`, `summarize`, `classify`, `serp_search`.
2. `call_provider(request)` — execute HTTP call with timeout and auth.
3. `validate_response(response)` — ensure expected fields or error codes.
4. `normalize_response(response)` — produce standardized `ProviderOutput` with fields & quality score.
5. `get_cost_estimate(response_meta)` — compute token/call estimate for billing.
6. `report_metrics(metrics)` — emit latency, success/failure, tokens.

ProviderOutput must include: `{content, extracted_fields?, quality_score, provider_meta:{name, model, latency_ms, attempt}}`

5. Search & Apply Flow (sequence)

```
sequenceDiagram
participant User
participant Orchestrator
participant Aggregator
participant ProviderMgr
participant InternalAI
participant JobsUnifiedDB
participant AppDB
```

```
User->>Orchestrator: Search(role, filters)
Orchestrator->>Aggregator: Plan query + use cache?
Aggregator->>JobsUnifiedDB: Check cache/index
JobsUnifiedDB-->>Aggregator: cached results? no
Aggregator->>ProviderMgr: call providers in parallel (SERP/APIs)
ProviderMgr->>ExternalProviders: calls...
ExternalProviders-->>ProviderMgr: raw results
ProviderMgr-->>Aggregator: enriched/normalized snippets
Aggregator->>InternalAI: parsing/enrichment (optional)
InternalAI-->>Aggregator: structured JobUnified
Aggregator->>Aggregator: dedupe & rank
Aggregator->>Orchestrator: return top N (with provider_meta)
Orchestrator->>User: show results (source badges, provider notes)
User->>Orchestrator: Apply(job_id)
Orchestrator->>AppDB: if OUR_BOARD -> show native apply form
Orchestrator->>User: show application form
User->>Orchestrator: Submit application
Orchestrator->>AppDB: store application + resume path
Orchestrator->>User: Confirmation (Application sent)
```

6. Model Training, Versioning & Replaceability

6.1 Model Registry

- Track each model: `model_id`, `type` (parser/chat/matcher), `version`, `trained_on_dataset_id`, `metrics` (precision/recall), `deployed` (yes/no), `deployed_at`.
- Promote new model versions via Admin Console.

6.2 Training Pipeline

- Inputs: labeled resumes, manual corrections (human-in-the-loop), synthetic augmentations.
- Process: data validation → train → evaluate → store metrics → push to model registry.
- Canary / A/B rollout: route subset of traffic to new model, compare metrics (parsing accuracy, latency, downstream conversion).

6.3 Replaceability

- Expose internal model as a microservice with standard API (same shape as provider adapter). Swap binary/model under the hood, no API change for callers.

- Keep backward compatibility by versioning endpoints: `/v1/parse`, `/v2/parse`.

---

7. Provider Rotation & Cost Controls

- **Policy controls**:
  - `weight` per provider.
  - `daily_limit` per provider (tokens/calls).
  - `cost_threshold` (total spend cap).
  - `quality_threshold` for fallback (if quality < X, demote provider).
- **Algorithms**:
  - Weighted round-robin + health checks.
  - Exponential backoff for retries, with jitter.
  - Circuit breaker: disable provider if error rate > threshold in window.
- **Admin ops**: immediate disable/enable, adjust weights, add API keys.

8. UI: How results appear to users

- Every job card shows:
  - Title, Company, Location, Excerpt
  - Badges: `Our Board` / `Company Site` / `Aggregator` / `LinkedIn` / `Google`
  - Provider note if fallback used: `via Together (fallback from OpenRouter)`
  - Buttons: `[Details] [Apply] [Save]`
- **Apply button behavior**:
  - `Our Board` → in-app native apply form (store in DB).
  - `External` → `Smart Prefill` if allowed + redirect to canonical apply_url; log click and offer "Mark as Applied".

9. Security, Compliance & Privacy

- PII encryption at rest and in transit.
- Access controls: RBAC for Admin / Recruiter / Candidate.
- Consent flows: show user that applying via external site may share info.
- Avoid scraping where TOS prohibits. Respect robots.txt and rate limits.
- Audit logs for provider calls and fallback chains (for transparency & billing).

10. Observability & SLA

- Metrics: per-provider latency, calls, tokens, cost, success rate; dedupe rate; cache hit rate; search query throughput.
- Alerts: provider error spike, cost overrun, model regression (accuracy drop).
- Tracing: propagate request-id across provider calls for debugging.

11. Admin Console Features (must-haves)

- Provider Manager UI (add/edit/delete): keys, weights, limits.
- Provider Health Dashboard: latency, errors, fallback counts.
- Model Registry: list models, metrics, promote/rollback.
- Search Analytics: #queries, top roles, conversion (apply click → application).
- Billing panel: provider spend & forecast.

12. Deployment & Scalability Notes

- Run AI microservices (internal models) on GPUs if fine-tuning / heavy inference; use autoscaling for LLM adapter workers.
- Use a message queue (Kafka/RabbitMQ) for async enrichment and heavy provider tasks.
- Use a search index (Elasticsearch/Opensearch) for JobsUnified for fast faceted search.
- CDN + object storage for resume downloads.

13. Minimal Implementation Roadmap (priority)

1. Build Orchestrator + AppDB + basic job board (our jobs CRUD, candidate apply).
2. Internal AI: simple parser service (rule-based + light ML) for resume -> structured.
3. Provider Manager + single external provider (OpenRouter/ChatGPT) for enrichment.
4. Aggregation Engine with 2–3 adapters (your DB + one aggregator + company feed).

5. Normalizer, dedupe, JobsUnified index.
6. Admin Console for provider config + metrics.
7. Training pipeline + model registry.
8. Add more providers, caching, cost controls, and canary model rollouts.

14. Example Operational Scenarios (short)

- **High-quality parse needed**: route to internal model v2 (GPU) for production parsing.
- **If internal overloaded**: ProviderMgr calls external model as temporary fallback (but avoid sending raw PII to third parties unless consented).
- **Search spikes**: increase cache TTL, autoscale adapter workers.