# Claude Code Prompt for Elara URL Scan Platform V2 Upgrade

Below is a detailed step‑by‑step prompt for Claude Code to upgrade the existing Elara platform (v1) to the new v2 architecture. This prompt assumes you have cloned the Elara‑Tanmoy/elara‑mvp‑final repository and are working on a **feature branch** off of dev. The goal is to **extend** the existing codebase without breaking current functionality while incorporating deterministic risk scoring, calibrated consensus, Vertex AI training, a Central AI/B2B API, and enhanced admin capabilities. All changes must be fully documented, tested and deployed to the GCP dev environment when finished.

## 0. Preparation

1. **Create a new branch:** Checkout from the latest dev branch and create a branch named feature/v2-scan-engine.
2. **Review the design document:** Familiarise yourself with the high‑level design in docs/architecture/elara\_v2\_final\_design.md (provided separately), which contains the target architecture diagram and implementation notes.
3. **Maintain backward compatibility:** All existing v1 scan endpoints, message/file scanning, deepfake detection, profile analyser, fact checker, recovery support, literacy coach and multi‑LLM verdicts **must continue to work**. v2 will be opt‑in via admin configuration.

## 1. URL Scanner V2 Module

### 1.1 Folder structure

Create a new folder packages/backend/src/scanners/url-scanner-v2/ with the following sub‑modules:

* **reachability.ts** – checks DNS/TCP/HTTP and classifies the URL as ONLINE, OFFLINE, WAF, PARKED or SINKHOLE. Return a ReachabilityResult with status and details.
* **evidence.ts** – uses a headless browser (existing proxyService) to collect HTML/DOM, HAR, forms, redirect chain, cookies/localStorage, TLS/WHOIS/DNS/ASN and a screenshot. Extract fields like form actions, auto‑download flags and script obfuscation.
* **feature-extract.ts** – converts raw evidence into structured features: lexical tokens (e.g. char n‑grams, entropy), tabular metrics (domain age, TLD/ASN buckets, TI hit counts, TLS anomalies) and causal signals (formOriginMismatch, brandInfraDivergence, redirectHomoglyphDelta, autoDownload). Reuse or extend existing category code where possible.
* **stage1.ts** – implements the lightweight models:
* **urlLexicalA**: existing XGBoost model for char n‑grams.
* **urlLexicalB**: new URLBERT/PhishBERT transformer. Load the model via @/models or call a Vertex AI endpoint. Fine‑tune the model as described in the training section.
* **tabularRisk**: monotonic XGBoost trained on tabular metrics and TI features. Use Vertex AI Feature Store for feature lookup.
* **stage2.ts** – invoked only when Stage‑1 is uncertain. Contains:
* **textPersuasion**: calls your fine‑tuned Gemma/Mixtral model via Vertex AI endpoint. Input: aggregated visible text, alt/title attributes and script hints. Output: probability of phishing/social engineering.
* **screenshotCnn**: calls your EfficientNet/ViT model via Vertex AI (GPU). Input: screenshot, output: probability of fake login or brand misuse.
* **combiner.ts** – fuses Stage‑1 and Stage‑2 logits with causal signals into a calibrated probability using a small logistic/GBM model. Calls conformal calibration code to produce a confidence interval. Accept a branch parameter (ONLINE, OFFLINE, etc.) to apply branch‑specific calibration thresholds.
* **policy.ts** – applies hard rules after the combiner:
* tombstone or sinkhole → block.
* dualTier1Hits (two or more top TI feeds within 7 days) → block.
* formOriginMismatch && brandInfraDivergence && domainAge < 30 days → high/critical.
* **index.ts** – orchestrates the v2 scan flow. It receives a URL, canonicalises it, checks the TI gate, runs reachability and evidence, calls Stage‑1, gates Stage‑2, runs the combiner and policy, and returns a structured EnhancedScanResult containing riskScore (probability × 100), riskLevel (A–F), confidenceInterval, decisionGraph, recommendedActions, screenshotUrl and skippedChecks.

### 1.2 Integration with existing code

1. Modify packages/backend/src/services/queue/scan.processor.ts to read a new field scanEngineVersion from the job payload. If v2, call url-scanner-v2; otherwise call the existing enhancedScanURL() (v1). When creating scan jobs from the API, default to v1 unless the admin has enabled v2.
2. Update packages/backend/src/controllers/scan.controller.ts to accept an optional query param ?version=v2 and pass it to the queue. Adjust validation accordingly.
3. Keep the return schema unchanged for compatibility (riskScore, riskLevel, verdict, etc.), but include additional fields (probability, confidenceInterval, decisionGraph) when using v2.

## 2. Consensus Service & Gemini Router

1. Create packages/backend/src/services/consensus/consensus.service.ts that accepts Stage‑1/Stage‑2 logits and causal signals, calls combiner.ts, applies calibration and policy, and returns the final verdict object. Provide helper functions to generate the decisionGraph and recommendedActions.
2. Create packages/backend/src/services/ai/geminiRouter.service.ts which abstracts calls to Vertex AI’s Gemini 1.5 Flash and Gemini 1.5 Pro models. Implement smart routing:
3. Use Flash (projects/.../publishers/google/models/gemini-1.5-flash) for simple summarisation tasks (scan verdict explanation, short answers).
4. Use Pro (.../gemini-1.5-pro) for complex analysis, multi‑step reasoning or long responses.
5. Implement response caching keyed by a hash of the prompt to avoid repeated calls.
6. Update the existing ai.service.ts to call geminiRouter.service.ts instead of multi‑LLM consensus. Preserve fallback to Claude/GPT if Vertex endpoints fail.

## 3. Training & Model Deployment

1. **BigQuery & Feature Store:** Create or update a bigquery dataset elara\_features\_v2 with tables to store scan features, TI hits and manual uploads. Use elara\_feature\_store (Vertex AI Feature Store) to serve slow‑moving features like domain age, ASN reputation and TI counts to both training and inference[[1]](https://cloudchipr.com/blog/vertex-ai#:~:text=Vertex%20AI%20is%20Google%20Cloud%E2%80%99s,using%20Google%20Cloud%27s%20robust%20infrastructure).
2. **Pipelines:** Implement Vertex AI pipelines (Jupyter or Cloud Build) that:
3. **Ingest** data from BigQuery, join with uploaded training data and derive features.
4. **Train** each model (URL lexical, tabular, text, image, combiner) using custom training jobs. Support hyperparameter tuning via Vertex AI’s HyperTune. For URL models, pretrain a URLBERT or PhishBERT encoder; fine‑tune on your labelled URL data.
5. **Calibrate** models using conformal prediction; output calibration thresholds per branch.
6. **Register** models in the Vertex Model Registry with metadata (version, date, metrics).
7. **Deploy** models to endpoints (CPU for Stage‑1, GPU for Stage‑2) with autoscaling.
8. **Monitor** for drift and trigger retraining when necessary. Use Vertex AI’s monitoring & explainability features[[2]](https://cloudchipr.com/blog/vertex-ai#:~:text=Beyond%20training%2C%20Vertex%20AI%20offers,MLOps%20features%20to%20operationalize%20AI).
9. **Data ingestion API:** Provide an admin API endpoint /api/v2/training/data/upload to upload CSV/XLSX/JSON/SQL data. Validate and write to BigQuery. Extend the admin UI to manage uploaded datasets and trigger training jobs.

## 4. Central AI & B2B APIs

1. **Unified AI endpoints:** Add a new router packages/backend/src/routes/central-ai.routes.ts exposing /api/v2/ai/analyze, /api/v2/ai/chat and /api/v2/scan/uri for B2B clients. These endpoints will:
2. Accept an API key or OAuth2 token.
3. Enforce per‑tenant quotas and rate limits.
4. For /ai/analyze, accept a URL, file or message, run the v2 scan pipeline, and return the verdict plus a Gemini explanation.
5. For /ai/chat, call the Ask Elara chatbot with optional context or scan ID and return a Gemini‑generated conversation using retrieval‑augmented generation.
6. For /scan/uri, allow direct v2 scanning with options to enable/disable modules (e.g. deepfake, profile, fact checker).
7. **Tenant isolation:** Use middleware to inject tenant ID into BigQuery queries and restrict data access to that tenant’s records.
8. **Billing & analytics:** Implement event logging for each API call; write usage stats to BigQuery for billing and dashboards.

## 5. Admin UI Modifications

1. **Scanning engine toggle:** Add a drop‑down in the Scan Engine Admin panel to select v1 or v2 for URL scans, defaulting to v1. Persist the selection in your configuration DB.
2. **Training dashboard:** Add pages to view datasets, schedule training jobs, view pipeline runs and model versions. Allow admins to launch new training runs with custom parameters (e.g. learning rate, number of epochs).
3. **API management:** Add UI to create API keys for B2B clients, set rate limits and see usage statistics.
4. **Mermaid diagrams:** Update docs/architecture/system-design.md and related files to reflect the new v2 flow and Central AI API. Use the Mermaid syntax to illustrate the request path from API Gateway → Request Router → Scan Engine v2 → Consensus Service → Data Lake/Feature Store → Vertex AI pipelines.

## 6. Testing & Deployment

1. **Unit tests:** Write comprehensive tests for each new module (reachability, evidence, feature extraction, stage‑1/2 models, combiner, policy). Mock external services (Vertex endpoints, BigQuery) to verify logic.
2. **Integration tests:** Simulate full v2 scans end‑to‑end on sample URLs (benign, phishing, scam). Validate risk scores, levels, decision graphs and Gemini summaries.
3. **CI/CD:** Update GitHub Actions to run the new tests and build your new modules. Use Terraform or Helm to deploy the new services to your dev GCP environment. Ensure environment variables (Vertex model endpoints, BigQuery dataset IDs) are configurable via Secrets Manager.
4. **Feature flags:** Wrap v2 features behind a feature flag so they can be rolled out gradually.
5. **Rollback plan:** Keep v1 code intact. Deploy v2 side‑by‑side and monitor metrics (precision, false positives, latency). If regressions occur, roll back by switching the engine selection back to v1.

## 7. Documentation & Change Management

1. **Update docs:** Add docs/architecture/v2/ describing the new modules, training pipelines and APIs. Include the ASCII diagram from the design document and explain how each component interacts with the rest of the system.
2. **Mermaid updates:** Locate existing Mermaid diagrams in docs/architecture/\*.md and update them to match the v2 design. Ensure the diagrams are committed.
3. **Changelog:** Append a section to CHANGELOG.md summarising the v2 upgrade and its features.
4. **README:** Add a high‑level overview of v2, its benefits and instructions for enabling it. Provide links to the training API docs.

## 8. Post‑Deployment Verification

1. After deploying to dev, run test scans on known malicious and benign URLs. Verify that the v2 engine returns correct risk levels and explanations.
2. Confirm that the Central AI and B2B API endpoints work for multiple tenants with enforced rate limits.
3. Validate that training jobs can be triggered from the admin UI and complete successfully, updating the model registry.
4. Monitor logs and metrics for any unexpected errors or performance issues.

## Important Notes

* **Vertex AI features:** Leverage Vertex AI’s unified platform for data preparation, training and model deployment[[1]](https://cloudchipr.com/blog/vertex-ai#:~:text=Vertex%20AI%20is%20Google%20Cloud%E2%80%99s,using%20Google%20Cloud%27s%20robust%20infrastructure). Use its **AutoML** and **custom training** to train models on your dataset[[3]](https://cloudchipr.com/blog/vertex-ai#:~:text=Vertex%20AI%20lets%20you%20do,users%20of%20all%20skill%20levels). Utilise **pipelines**, **feature store** and **model registry** to support MLOps[[2]](https://cloudchipr.com/blog/vertex-ai#:~:text=Beyond%20training%2C%20Vertex%20AI%20offers,MLOps%20features%20to%20operationalize%20AI).
* **Security & privacy:** Do not store any user PII in logs or training data. Hash sensitive fields (e.g. URLs, IPs) before storage. Ensure OAuth scopes and IAM roles follow the principle of least privilege.
* **Accuracy vs. completeness:** You cannot guarantee 100 % accuracy. Use calibrated probabilities and confidence intervals to communicate uncertainty. Provide clear recommendations and highlight when human review is required.

## Summary

Follow these steps to integrate the v2 scanning engine, Vertex AI models, Gemini‑based reasoning and B2B APIs into the existing Elara platform. The modifications must be incremental, non‑breaking and fully documented. Once complete, commit your changes to the feature/v2-scan-engine branch, open a pull request, and tag the appropriate reviewers.

[[1]](https://cloudchipr.com/blog/vertex-ai#:~:text=Vertex%20AI%20is%20Google%20Cloud%E2%80%99s,using%20Google%20Cloud%27s%20robust%20infrastructure) [[2]](https://cloudchipr.com/blog/vertex-ai#:~:text=Beyond%20training%2C%20Vertex%20AI%20offers,MLOps%20features%20to%20operationalize%20AI) [[3]](https://cloudchipr.com/blog/vertex-ai#:~:text=Vertex%20AI%20lets%20you%20do,users%20of%20all%20skill%20levels) What Is Vertex AI? Streamlining ML Workflows on Google Cloud

<https://cloudchipr.com/blog/vertex-ai>