QuantumAI — Prototype Development Plan

Purpose

This document is the canonical project plan and repo skeleton for the first development phase of the Quantum–Classical Hybrid AI Platform. It focuses on delivering the highest-impact enterprise primitives first:

1. **Crypto-Agility Reference Implementation** (Priority A)
2. **Error Mitigation (EM) Policy Engine & Nightly Regression Harness** (Priority B)
3. **Noise-/EM-aware Scheduler v1** (Priority B)

Each deliverable is laid out with technical design, repo structure, sprints, acceptance criteria, CI strategy, and required access.

High-level Goals (MVP scope)

- Ship an HSM-backed KMS demo that supports dual-stack TLS (classical + PQC) and signed run artifacts.
- Build a reproducible EM benchmarking harness that runs ML-QEM and ZNE variants across simulators and available backends, and outputs a nightly dashboard.
- Implement a scheduler plugin that uses backend calibration data + EM cost estimates to route jobs to the cheapest backend meeting an accuracy SLA.

Timeline & Milestones

Sprint 0 — Setup (3 days)

- Create repo, CI, basic infra templates (K8s manifests), developer onboarding docs.

Sprint 1 — Crypto-Agility Prototype (2 weeks)

- HSM-backed KMS + API, dual-stack TLS demo, signed run-manifest generator (CLI + SDK hook), migration playbook draft.

Sprint 2 — Nightly EM Regression Harness (2.5 weeks)

- Create harness to run set of circuit classes in simulator and selected vendor backends, implement ML-QEM baseline + ZNE baseline, nightly runner + dashboard.

Sprint 3 — EM Policy Engine Prototype (2 weeks)

- Train selector (random forest + small NN ensemble) to pick EM method, integrate with nightly results, provide REST endpoint.

Sprint 4 — Noise/EM-aware Scheduler v1 (2 weeks)

- Implement scoring heuristics, routing plugin, integration tests, encryption/crypto-mode enforcement logic.

Buffer & Documentation (1 week)

- Finalize docs, create demo playbook, record walk-through videos or notebooks.

Total elapsed: \~10–11 weeks for polished prototypes (can be compressed with more engineers). Priority A features (crypto baseline and manifest) will be usable earlier (end of Sprint 1).

Repo Layout (monorepo recommended)

```
quantumai/
├─ infra/                  # Terraform + Helm charts + K8s manifests
├─ api/                    # Orchestration API (gRPC + REST)
├─ sdk/                    # Python SDK (client, manifest hooks)
├─ cli/                    # CLI tools (manifest generator, job submit)
├─ kms/                    # KMS & HSM integration (service + tests)
├─ em-harness/             # Nightly EM regression harness + datasets
├─ em-policy/              # EM policy engine model & service
├─ scheduler/              # Scheduler service + plugins
├─ ui/                     # Grafana dashboards + minimal React UI for sandbox
├─ notebooks/              # Example notebooks (PennyLane/Qiskit examples)
├─ tests/                  # Integration & e2e tests
└─ docs/                   # Design docs + playbooks
```

Detailed Deliverable Designs

A. Crypto-Agility Reference Implementation

Objective: HSM-backed KMS that supports classical keys and PQC keys (FIPS 203/204/205), dual-stack TLS demo, and signed run-manifests.

Key features & components

- HSM integration module (PKCS#11 or cloud KMS with HSM-backed keys).

- Key types: classical RSA/ECDSA (for backward compatibility) + ML-KEM (Kyber-like) + ML-DSA (Dilithium-like) + SPHINCS+ for long-term signatures.
- Dual-stack TLS demo: TLS handshake that negotiates classical cipher + PQC KEM in parallel (demo mode), and fallback.
- Run-manifest generator: JSON/YAML manifest with fields for PQC suite, backend id, calibration snapshot id, EM method/version, seeds, and artifact signature (signed with HSM private key).
- Migration playbook: inventory, cutover steps, dual-write/dual-verify migration plan, deprecation timeline.

Acceptance criteria

- KMS can generate/rotate PQC keys and export public keys.
- CLI can produce signed manifests and verify signatures.
- Demo client/server can negotiate dual-stack TLS and log selected PQC suite.
- Playbook provides concrete steps and sample commands for migration.

Security

- HSM usage enforced; secrets never land in plaintext in storage.
- Rotation scripts and audit logs in place.

B. EM Regression Harness & ML-QEM Baseline

Objective: Reproducible test harness that runs predefined circuit families and compares EM methods.

Circuit families

- Random circuits (varying depth & qubit count)
- Trotterized dynamics circuits (small molecules)
- Variational circuits (VQE/QAOA-pattern)

Methods implemented

- ZNE (Zero Noise Extrapolation) baseline
- Richardson extrapolation variants
- ML-QEM baseline (random forest predictor trained to correct measurement errors)

Outputs

- Per-circuit accuracy (error w.r.t simulator/noiseless baseline) vs runtime and shot count
- Comparative charts for accuracy vs cost
- Data persisted (Parquet) with metadata for lineage

Acceptance criteria

- Harness can run on simulator locally and submit jobs to at least one cloud QPU (or mock adapter) in CI.

- Nightly pipeline runs and writes a dashboard data snapshot.
- Baseline ML-QEM model shows improvement on at least two circuit families vs unmitigated runs in simulation.

C. EM Policy Engine

Objective: Policy/ML service that recommends an EM method and estimates overhead and expected post-EM fidelity.

Inputs

- Circuit features (depth, entangling gate count, qubit count)
- Backend noise profile (gate fidelities, readout error rates)
- Shot budget & latency constraint
- Past EM performance (historical dataset from harness)

Outputs

- Recommended EM method + parameters
- Estimated multiplicative shot overhead and expected fidelity
- Confidence score

Model & Tech

- Feature engineering module (scikit-learn pipeline)
- Ensemble model: random forest for ranking + small NN for regression of overhead
- REST service (FastAPI) + metrics export

Acceptance criteria

- Policy correctly ranks EM choices in >X% of offline test cases (configurable, e.g., 80%)
- Service latency <200ms for decision on typical circuit features

---

D. Noise/EM-aware Scheduler v1

Objective: Routing layer that scores backends by expected post-EM accuracy vs cost and routes jobs accordingly; enforces `crypto-mode` constraints.

Scoring heuristic (v1) score = (expected_postEM_fidelity / cost) * availability_penalty * crypto_compatibility

Features

- Enforce crypto-mode: if tenant requires PQC or QKD, scheduler only picks compatible backends & network routes
- Persist routing decisions to signed manifests

- Integrate with em-policy to request EM overhead estimates

Acceptance criteria

- Scheduler chooses backends matching offline oracle in >X% of cases for test workload mix
- Crypto-mode blocks incompatible backends and logs reasons in manifest

---

CI/CD Strategy

- Use GitHub Actions / GitLab CI for PR checks
- Unit tests + blackbox integration tests using mock QPU adapters
- Nightly integration test job for EM harness (runs subset of circuits)
- Release pipeline produces docker images with signatures (dual-signed release artifacts)

Required Access & Dependencies (initial)

- Cloud account(s) for simulator & QPU vendor sandbox (IBM/Braket/Azure) or vendor API keys
- HSM (cloud HSM like AWS CloudHSM / Azure Key Vault HSM) or test PKCS#11 simulator (for dev)
- Grafana / Prometheus stack for dashboards (dev sandbox)
- Container registry for images

Security & Compliance Checklist (minimum)

- HSM-backed key storage & rotation policies
- Signed manifests for provenance (HSM)
- Dual-stack TLS demo to show PQC negotiation
- Access logging & SIEM integration for critical control plane actions

---

Deliverables per Sprint (summary)

- Sprint 1: HSM KMS service + manifest CLI + migration playbook + tests
- Sprint 2: EM harness + ML-QEM baseline + nightly runner + dashboard
- Sprint 3: EM policy engine service + evaluation + REST endpoint
- Sprint 4: Scheduler plugin + crypto-mode enforcement + integration tests

---

Example commands (dev quickstart)

```
# clone
git clone git@github.com:yourorg/quantumai.git
```

```
cd quantumai

# Start dev K8s (kind/minikube)
make infra/dev-up

# Run the KMS service (dev HSM simulator)
cd kms && pip install -r requirements.txt && uvicorn kms.main:app --reload

# Generate a signed manifest
python cli/generate_manifest.py --job-id 123 --backend ibm_qpu_1 --pqc-suite ML-
KEM:ML-DSA --em-method ml-qem

# Run EM harness example (simulator)
cd em-harness && python run_example.py --circuit random --depth 6
```

Reporting & Dashboards

- Grafana dashboards for EM nightly results and scheduler metrics
- PDF/HTML exportable run reports (signed manifests + metric snapshot) for audits

---

Next immediate steps (today)

1. Confirm priorities & team size (do we staff 1 or 2 engineers per stream?).
2. Provision dev resources (cloud keys, HSM dev or simulator, Grafana).
3. I will scaffold the repo with the directory structure and initial README + Sprint 0 automation (CI skeleton & Makefile).

Who will do what (roles suggested)

- Lead engineer (you or appointed) — architecture owner & code reviews
- KMS/HSM engineer — implement HSM integration and manifest signing
- Quantum infra engineer — EM harness + vendor adapters
- ML engineer — ML-QEM baseline + EM policy model
- Scheduler/devops — scheduler plugin + cost heuristics + K8s infra
- QA — nightly CI & regression suite validation

Acceptance & Handover

At the end of Sprint 4, we will provide:

- Working repo with CI & tests
- Demo playbook reproducing key flows (signed manifest, EM harness runs, scheduler routing)
- Documentation & migration playbook
- A final technical report summarizing results vs acceptance criteria

---

If you confirm, I will scaffold the repository and push initial commits for Sprint 0 (repo skeleton, CI, infra templates, README). Please confirm team size and whether you want cloud vendor sandboxes (IBM/Braket/Azure) provisioned now or later.

confirmed