QAI_LLM_DevOps_Framework_Notes

QAI LLMs based on various class types, taxonomies, Industrial job class types, ToDo Tasks of human day to day personal, office, social, and various occasions like parties, anniversaries, tours, kids, etc.

These LLMs need to be developed inhouse in a limited CPU, NPU, TPU processor capacity , may be some cloud hosted hardware.

These LLMs can be general purpose or specific to a class or a problem- real world, computing works, CPS world, Robot World etc

These LLMs need to use various emerging technologies and models like the OpenAI models, hybrid multimodal models, and our home grown products like the QAI processor, QAI OS, QAI Datacenter etc. Many more data structures, functions, algorithms like use of the QAI RAG, QAI Vector DB, QAI algorithms, QAI Hybrid solvers, QAI data processors, QAI OpsAgent, full scale QAI stack along with various GUI, block chain based, Post quantum cryptography etc

These QAI LLMs need to work with our home grown frameworks like the Org framework, 16 Ops frameworks, One Business transformation framework, Business compliance tools, Apps, etc

QAI LLMs need to work as a standalone, edge or inference modes for various model configurations in a standalone or distribute computing configuration.

QAI LLMs can take data from Gen AI data sets, feeds and also develop various dynamic ways to pick the right data sensibly from a ETL, ELT, Data warehouse, Star schemas for various purposes like MIS report, Time series analysis, forecasting based on historical project data etc

QAI LLMs can have various form factors allowing them to work in various constraints like low memory footprints, low power, low CPU power, or isolated places, air gapped systems, Robotic parts and arms, In space, under water, etc

In summary QAI LLMs can now be made more of an integrated hybrid embedded system inside a larger system subject to Software and Systems Engineering standards, and adhere to the external systems configurations, commands and relay required data, cooperate and work autonomously

This LLMs need to adhere to AI ethics and provide safety first based services.

These can work in a single mode or Agentic AI crew modes and offer various types of quality of services, real time, offline batched operations like an earlier mainframe that used to process accounts and payrolls.

These LLM can adjust to various dynamic change requests like upgrading to newer standard version, newer standard, dynamically changing data, motion, ambient condition etc that can help users to test in remote and un reachable spaces like on the space shuttle, inside space research labs, under water, inside Earth, on the Moon etc

With this type of LLM architecture, this can be a Go-To Tool for many multi roles, multi problems.

As a startup, these LLMs can help the owners to make good business, suggest various techniques, increase product quality etc. Similar roles can be played for various target environments

Details:

1 — Short summary (one sentence)

Design a family of **QAI_LLMs** (general + class-specific) as hybrid, modular LLM subsystems that run on constrained local hardware or cloud, integrate tightly with your QAI processor/OS/datacenter and Org/Business frameworks, support agentic modes & RAG, and are governed by ethics, security, and software/system engineering standards.

2 — High-level architecture (modular, hybrid)

1. **Model layer**
   - Family of LLMs: micro-LMs (tiny), edge-LMs (small/quantized), mid/large models (cloud/QPU-assisted).
   - Modality adapters: text, vision, audio, sensor streams, task-specific encoders for robotics/CPS.
   - Agent layer: orchestration of skills (tool use, RAG, planners, executors).
2. **Data & retrieval layer**
   - QAI Vector DB + hybrid RAG indexers (local + distributed).
   - ETL/ELT connectors: warehouse, star schemas, time-series stores, HIL (human-in-the-loop) labeling.
3. **Runtime & infra layer**
   - Local runtime (QAI OS): optimized kernels for QAI processor / NPU / accelerator.
   - Inference serving: multi-mode (standalone, cluster, air-gapped).
   - Edge orchestration: lightweight orchestrator (k3s/nomad-like), plus offline batch scheduler.
4. **Ops & governance**
   - QAI OpsAgent: CI/CD, policy enforcement, auditing, model registry, artifact signing (post-quantum-ready).
   - Observability: telemetry for latency, accuracy, drift, resource usage.
5. **Security & compliance**
   - Data encryption (at-rest & in-flight), attestation, secure boot, supply-chain protection, PQC for sensitive flows.
   - Role-based access + audit logs.

3 — Class types & taxonomies (how to organise many LLM variants)

Organize by *purpose* + *deployment footprint* + *domain*:

A. By purpose

- Personal assistant (day-to-day tasks, calendar, reminders)
- Office productivity (email summarization, MIS, forecasting)
- Social/Urgent (replies for parties, anniversaries, personal suggestions)
- Industrial/Robot/CPS (control loops, command translation, safety checks)
- Scientific/Space/Underwater (domain-specific solvers, telemetry summarizers)
- Compliance & Legal (policy guidance, regulatory checks)

B. By footprint

- Tiny (<= 50M params, on-device, real-time)
- Small (50–500M, edge NPU)
- Medium (500M–4B, local multi-core/NPU)
- Large (>4B, cloud or QPU-accelerated)

C. By modality

- Text-only
- Multimodal (text+vision+sensor)
- Embedded (binary/procedural interfaces for robotics)

4 — Data architecture & pipelines

- **Data sources:** sensor streams (robot joints, IMU), application logs, historical project DB (star schema), gen-AI corpora, user uploads.
- **Ingestion:** connectors for batch (ELT) & streaming (Kafka/GCP PubSub) with pre-processing.
- **Storage:** cold (data lake), warm (warehouse/OLAP for MIS), fast (vector DB for RAG), time-series DB for telemetry.
- **Labeling / HIL:** tools to collect corrections, create reward signals, and maintain a labeled dataset registry.
- **Privacy:** PII scrubbers, differential privacy options for aggregated training updates.

5 — Model training & optimization for limited hardware

1. **Model selection & topology**
    - Use parameter- and compute-efficient architectures (ex: quantized transformers, MoE sparingly, distilled variants).
2. **Compression techniques**
    - Distillation → train small student models from larger teachers.
    - Quantization (8-bit, 4-bit, or integer-only inference).
    - Pruning + structured sparsity for NPU-friendly ops.
3. **Training strategies**

    - Federated/edge fine-tuning (when data is sensitive); central aggregation on QAI datacenter.
    - LoRA / adapters for task-specific fine-tuning (cheap and fast).
    - Hybrid training: heavy training on cloud/QPU, edge fine-tune via adapters.
4. **Hardware-aware compilation**
    - Compile kernels specifically for QAI processor (fused ops, memory tiling).
    - Autotune runtime parameters (batch size, thread pinning).
5. **Efficient inference**
    - Dynamic precision (mixed-precision), early-exit heads for faster answers.
    - Cache embeddings/answers for repeated prompts.

6 — RAG, Vector DB & knowledge management

- Maintain hybrid indices: local vector cache (on-device), synced with central QAI Vector DB.
- Smart retrieval policies:
    - Short-term cache for recent contexts
    - Long-range knowledge retrieval for domain facts
    - Privacy labeling: some knowledge is anonymized or restricted
- Versioned knowledge bundles (signed, verifiable; good for air-gapped or space deployments)

7 — Serving & deployment modes

- **Modes**
    - Real-time interactive (low latency, tiny/small models)
    - Batch/offline (reporting, payroll-style jobs)
    - Agentic crew (multiple agents cooperating — planner, exec, guardrails)
- **Form factors**
    - On-device air-gapped (signed bundles + local inferencer)
    - Edge cluster (multi-node with orchestrator)
    - Cloud/QPU-assisted (heavy models and centralized aggregation)
- **Edge resilience**
    - Graceful degradation (fallback to simpler policies / cached replies)
    - Telemetry sync when network available

8 — DevOps & lifecycle (QAI LLM DevOps Framework)

1. **Model registry**: versioned models + adapters, artifacts signed, model cards attached.

2. **CI/CD for models**
    - Data tests (schema, statistical checks)
    - Training pipeline (unit test on model training components)
    - Continuous evaluation (automated tests vs. validation sets + safety checks)
    - Deployment pipeline: canary -> staged rollout -> full
3. **Infra as code**
    - Declarative specs for runtime bundles, resource profiles (tiny/edge/medium/cloud)
4. **Policies & governance**
    - Policy-as-code for safety rules, data handling, and allowed tools
5. **Rollbacks & A/B**
    - Fast rollback by switching to earlier signed model artifact
6. **Observability**
    - Logs, traces, metrics (latency, throughput)
    - Quality telemetry: answer correctness, hallucination rate, user feedback
    - Drift detection: model/feature/data drift alerts

9 — Testing, validation & safety

- **Automated tests**
    - Unit tests for tokenizers, adapters, and tool connectors
    - Integration tests for RAG, Vector DB, and retrieval pipelines
- **Evaluation suites**
    - Domain-specific accuracy, robustness tests under sensor noise, adversarial prompts
    - Safety suite: prompt injections, privacy leakage, encryption correctness
- **Human-in-loop**
    - Labeling for high-risk decisions; approval gating for production updates
- **Red-team**
    - Internal offensive testing to find failure modes & hallucinations

10 — Security, compliance & PQC integration

- **Data governance**
    - PII detection & redaction
    - Data access governance with RBAC / ABAC
- **Artifact security**
    - Signed model bundles with post-quantum signatures (for long-lived devices like space)
    - Secure boot and attestation for QAI OS images

- **Communication**
  - TLS + PQC for long-term deployments where adversaries may have quantum resources
- **Air-gapped workflow**
  - Signed update packages, reproducible build manifests, and offline verification steps

11 — Ethics & explainability

- Model cards & datasheets for every released model and adapter.
- Explainability modules: local explanation (saliency, retrieval provenance), global explanation (policy rationale).
- Consent & opt-out flows for personal data.
- Safety-first fallbacks: if uncertain or safety-risk, defer to a human or safe canned response.

12 — Observability & SLOs (example metrics)

- Latency SLOs: p95 < X ms for tiny models; define tiers for each form factor.
- Availability: 99.9% for mission-critical robot controllers.
- Correctness: accuracy or task success rate thresholds per domain (MIS forecast: MAPE < Y%).
- Drift threshold: trigger retraining if performance drops Z% from baseline.

13 — Business model & go-to-market suggestions

- **Product lines**
  - On-prem licensed LLM bundles (signed, air-gap friendly) for industrial/CPS customers.
  - Edge subscription (OTA updates + vector sync) for SMB robotics.
  - Cloud + QPU premium (high-compute, heavy models).
  - Consulting & integration (tuning LLMs to customer workflows).
- **Monetization**
  - Per-device license + support
  - Model fine-tuning as a service
  - Data labeling + managed RAG indexing
- **Differentiators**
  - Tight QAI processor/OS integration -> lower power/latency
  - Industry-specific safety certifications & PQC readiness
  - Strong offline / air-gapped story (space, defense, underwater)

14 — Integration with your existing QAI ecosystem (concrete touches)

- QAI OS: deploy optimized runtime images and provide kernel-optimized operators.
- QAI Processor: build compiler backends and kernels for quantized ops + autotuning
- QAI Datacenter: centralized training, model registry, federated aggregation
- QAI Vector DB: unified retrieval layer with signed knowledge bundles
- QAI OpsAgent: orchestrate CI/CD, policy enforcement, telemetry forwarding
- Org & 16 Ops frameworks: map model capabilities into business process tasks and create automation templates (e.g., payroll processing agent, predictive maintenance agent)

15 — Prioritized first tasks (concrete checklist)

1. **Define taxonomy**: create a concise matrix of use-cases × model footprint × modality (document in repo).
2. **Create minimal model card template** for all LLMs (fields: purpose, footprint, SLOs, data sources, training dates, audit info, signature).
3. **Prototype tiny model** (Tiny assistant: ≤50M) for a representative consumer workflow (calendar + reminders) with RAG disabled — run on QAI OS/Python runtime.
4. **Set up model registry** (artifact storage + signed manifests).
5. **Implement a small vector DB proof** (local) and a retrieval pipeline with provenance metadata.
6. **Build CI pipeline skeleton**: data validation → training (toy) → eval → deploy to canary.
7. **Security baseline**: PII scrubber + basic artifact signing.
8. **Observability baseline**: collect latency, token count, success metric for the prototype.
9. **Safety policy draft**: top-10 banned/guarded actions and escalation rules.
10. **Plan a red-team session** schedule and criteria (when to run).

16 — Example artifacts I can produce next (pick any)

- One-page architecture diagram (SVG/PNG)
- Model card template (YAML + markdown)
- CI/CD pipeline YAML skeleton (GitHub Actions/GitLab/Argo)
- RAG retrieval policy pseudocode + vector DB schema
- Sample adapter/LoRA fine-tune script tuned for QAI processor
- Edge deployment manifest + signed update flow description
- Minimal canary rollout playbook with rollback commands

QAI_LLM_Framework_and_DevOps

This document is structured into two major parts:

1. **QAI LLM Architecture & Models** — design, taxonomy, and integration of QAI-specific LLMs.
2. **QAI LLM DevOps Lifecycle & Testing** — processes, compliance, testing, and operational lifecycle.

Part I — QAI LLM Architecture & Models

1. Short Summary (One Sentence)

Design a family of **QAI LLMs** (general + class-specific) as hybrid, modular LLM subsystems that run on constrained local hardware or cloud, integrate tightly with your QAI processor/OS/datacenter and Org/Business frameworks, support agentic modes & RAG, and are governed by ethics, security, and software/system engineering standards.

2. High-Level Architecture

1. **Model layer**
   - Family of LLMs: micro-LMs (tiny), edge-LMs (small/quantized), mid/large models (cloud/QPU-assisted).
   - Modality adapters: text, vision, audio, sensor streams, task-specific encoders for robotics/CPS.
   - Agent layer: orchestration of skills (tool use, RAG, planners, executors).
2. **Data & retrieval layer**
   - QAI Vector DB + hybrid RAG indexers (local + distributed).
   - ETL/ELT connectors: warehouse, star schemas, time-series stores, HIL (human-in-the-loop) labeling.
3. **Runtime & infra layer**
   - Local runtime (QAI OS): optimized kernels for QAI processor / NPU / accelerator.
   - Inference serving: multi-mode (standalone, cluster, air-gapped).
   - Edge orchestration: lightweight orchestrator, offline batch scheduler.
4. **Ops & governance**
   - QAI OpsAgent: CI/CD, policy enforcement, auditing, model registry, artifact signing.
   - Observability: telemetry for latency, accuracy, drift, resource usage.
5. **Security & compliance**
   - Encryption, secure boot, PQC for sensitive flows, RBAC + audit logs.

3. Class Types & Taxonomies

Organized by:

- **Purpose:** personal assistant, office productivity, social, industrial/CPS, scientific, compliance/legal.
- **Footprint:** Tiny (≤50M params), Small (50–500M), Medium (500M–4B), Large (>4B).
- **Modality:** text-only, multimodal, embedded robotics connectors.

4. Data Architecture & Pipelines

- Data sources: sensors, MIS DBs, time series, external GenAI datasets.
- Storage tiers: cold (lake), warm (warehouse), fast (vector DB), time-series.
- Labeling & privacy: HIL annotation, PII scrubbers, differential privacy.

5. Model Training & Optimization

- Compression: distillation, quantization, pruning.
- Training: centralized, federated, on-device continual.
- Hardware-aware: kernels tuned for QAI processor.

6. Serving & Deployment Modes

- Real-time, batch/offline, agentic crew.
- Form factors: on-device, edge, cloud.
- Resilience: graceful degradation, telemetry sync.

7. Ethics & Safety

- Model cards & datasheets.
- Explainability modules.
- Safety-first fallback strategies.

Part II — QAI LLM DevOps Lifecycle & Testing

This section defines the DevOps lifecycle, standards compliance, testing, troubleshooting, non-functional testing, configuration templates, training & release practices for the QAI LLM family. It is intended as a modular section that links back to requirements, taxonomy, QAI stack components and business frameworks.

Contents

1. DevOps lifecycle overview
2. Standards & compliance mapping
3. Artifacts & traceability
4. CI/CD pipeline (model + infra)
5. Training pipeline for QAI models
6. Non-functional testing & test catalog
7. Troubleshooting & incident runbooks
8. Templates & placeholders for dynamic configuration
9. Release windows, gating & rollout strategy
10. Inputs/Outputs taxonomy & interfaces
11. Agent/Quantum/Abstract class sketches
12. Observability, SLOs & metrics
13. Security hardening & PQC artifact flows
14. Appendix: YAMLs, checklists, templates

---

1. DevOps lifecycle overview

A linear + iterative lifecycle tailored for QAI LLMs. Stages map to traceable artifacts and are automated:

1. **Requirements & Taxonomy** → use-case, footprint, SLOs, safety, compliance. Artifacts: Requirement Spec, Threat Model, Model Card stub.
2. **Design & Ansätze** → model architecture, network topology, quantum circuit outlines, adapter/LoRA plan, data contracts.
3. **Data Engineering** → ETL/ELT, schema definitions, quality rules, labeling/annotation plans.
4. **Model Development** → experiments, reproducible runs, experiment tracking. Artifacts: weights, logs, evaluation reports.
5. **Validation & Testing** → unit, integration, regression, safety, adversarial/red-team, NFAs. Artifacts: test reports, benchmarks.
6. **Packaging & Signing** → signed, versioned bundles + PQC signatures. Artifacts: model card, datasheet, manifest.
7. **Canary/Deployment** → staged rollout, A/B, telemetry-gated.
8. **Monitoring & Feedback** → metrics, drift detection, user feedback ingestion.
9. **Maintenance & Retrain** → schedule retrains, adapter updates, security patches.
10. **Decommission** → retirement, artifact archive, key revocation.

---

2. Standards & compliance mapping

- **ISO/IEC 27001** → security mgmt (access control, key mgmt).
- **ISO/IEC 12207** → lifecycle processes.
- **IEEE 7000-series** → ethically aligned design.
- **NIST AI RMF** → risk management, governance.
- **NIST SP 800-53 / 207** → zero trust, supply chain.
- **IEC 61508 / ISO 26262** → safety-critical systems.
- **OWASP ASVS** → API hardening.
- **ISO/IEC 27701** → privacy info mgmt.
- **NIST PQC** → post-quantum cryptography.

Approach: maintain a **mapping table** → requirement → artifact/control → owner → verification.

---

3. Artifacts & traceability

Versioned + linked in a traceability matrix:

- Requirements spec
- Threat model, risk register
- Model card + datasheet
- Dataset manifests + provenance
- Training logs + hyperparameters
- Model artifacts (signed) + build manifests
- Test results (unit/NFA)
- Deployment manifests
- Observability configs + baselines
- Incident logs, postmortems

Traceability matrix columns: Requirement ID → Model(s) → Dataset(s) → Test(s) → Compliance Item → Owner → Status.

---

4. CI/CD pipeline (model + infra)

Model track:

- Data validation (schema, bias, PII).
- Preprocessing + feature tests.
- Train runs (reproducible).
- Evaluation (accuracy, safety, NFA smoke).
- Package signed bundles (weights + card).
- Publish to Model Registry.

Infra track:

- Build runtime image (QAI OS kernels).
- Unit + integration tests.
- Security scans (SBOM, vuln check).
- Bundle + sign.

Converge:

- Pre-release validation sandbox.
- Canary rollout → telemetry gating → full rollout.

---

5. Training pipeline for QAI models (classical + quantum-assisted)

- **Centralized** → full pretrain on QAI Datacenter + QPU acceleration.
- **Hybrid** → teacher models → distill into edge students.
- **Federated/Edge fine-tuning** → adapters updated locally, secure aggregation centrally.
- **On-device continual learning** → strict guardrails + privacy.

Steps: Dataset curation → Teacher runs → Distillation → Adapter training → QAT → Hardware autotuning.
Reproducibility: dataset hashes, seeds, hyperparameters, SBOMs.

---

6. Non-functional testing (NFA) & test catalog

- **Performance**: latency (p95), throughput, concurrency.
- **Reliability**: availability tests, chaos experiments.
- **Power/Thermal**: watts, throttling.
- **Robustness**: noise robustness, adversarial inputs, hallucination metrics.
- **Security**: PII leakage, pen-tests, SBOM validation.
- **Compliance & Safety**: ethics tests, fairness, content filters.
- **Environment**: airgap workflows, embedded stress tests.

Automation: integrate NFAs in CI, fail gates if regression > thresholds.

---

7. Troubleshooting & incident runbooks

Runbook A: High latency

- Symptom: p95 spike.
- Quick fix: divert traffic, fallback to smaller model.
- Root cause: logs, kernel counters.
- Rollback: revert signed bundle.

Runbook B: Unsafe output

- Symptom: policy violation.
- Mitigation: block output, human approval queue.
- Root cause: red-team reproduction, poisoned RAG.

Runbook C: Drift detected

- Symptom: quality drop.
- Mitigation: pin older model, pause retrain.
- Root cause: data shift.

All runbooks → postmortem template: timeline, impact, root cause, remediation.

---

8. Templates & placeholders for dynamic configuration

ModelBundle YAML (example)

```
apiVersion: qai/v1
kind: ModelBundle
metadata:
  name: personal-assistant-tiny
  version: 1.2.0
spec:
  model:
    architecture: transformer
    params: 45M
    quantization: int8
  runtimeProfile: tiny-edge-v1
  inputs: [text, calendar_events]
  outputs: [text, actions]
  safety:
    policySet: default-safety-v2
```

RuntimeProfile YAML → CPU, NPU memory, power caps, network mode.
Placeholders: `${ENV}`, `${HARDWARE_PROFILE}`, `${DATASET_HASH}`.

---

9. Release windows, gating & rollout strategy

- **Patch**: weekly, small bug/security.
- **Feature**: monthly, new adapters.
- **Major**: quarterly, new base model.

Gating: data validation → security scans → tests → NFA smoke → canary → rollout.
Blackout windows: e.g., during mission ops.
Emergency patch: minimal, postmortem mandatory.

---

10. Inputs/Outputs taxonomy & interfaces

Inputs: text prompts, structured JSON, time-series, sensor streams, binary media, agent messages, quantum circuits.
Outputs: text, action envelopes, telemetry summaries, multimodal bundles, signed proof bundles.

Contract: every I/O includes provenance, classification, confidence.

---

11. Agent / Quantum / Abstract class sketches

```
class InputAdapter(ABC):
    def parse(self, raw) -> CanonicalInput: ...
    def validate(self, canonical: CanonicalInput) -> bool: ...

class ModelInterface(ABC):
    def infer(self, input: CanonicalInput, config: ModelConfig) -> CanonicalOutput: ...
    def explain(self, input: CanonicalInput) -> Explanation: ...

class Agent(ABC):
    def plan(self, goal, context) -> Plan: ...
    def execute(self, plan: Plan) -> Result: ...
    def monitor(self, result) -> Observation: ...

class QuantumModule(ABC):
    def compile_ansatz(self, spec) -> QPUJob: ...
    def run_qpu(self, job: QPUJob) -> QPUResult: ...
    def postprocess_qpu(self, result: QPUResult) -> Tensor: ...
```

---

12. Observability, SLOs & metrics

Metrics:

- `qai_infer_latency_ms{model}`
- `qai_success_rate{task,model}`
- `qai_drift_score{model}`
- `qai_power_draw_watts{node}`
- `qai_safety_incidents_total{model}`

SLOs:

- Edge tiny model: p95 < 150ms.
- Robotics control: deterministic <30ms.
- Large API: p95 < 1s, >100rps throughput.

Alerts: latency >1.5x, safety incident >0, drift > threshold.

---

13. Security hardening & PQC artifact flows

- SBOMs for builds, vuln scans.
- PQC-capable signatures for bundles.
- Secure boot, attestation, reproducibility.
- Federated updates with secure aggregation + DP.

---

14. Appendix: sample YAMLs, checklists, and templates

□ Model Card Template (Markdown / YAML hybrid)

```
model_name: personal-assistant-tiny
version: 1.2.0
owners:
  - name: QAI Team
    contact: support@qai.example
purpose: >
  Daily task management and personal reminders
intended_use:
  - Personal productivity
  - Calendar management
not_intended_use:
  - Medical, legal, or safety-critical tasks
slos:
  latency_p95_ms: 150
  availability: "99.9%"
datasets:
  - name: calendar_corpus_v3
    source: QAI Datacenter
    hash: abc123...
```

```
training:
  date: 2025-09-01
  methods: ["distillation", "quantization-aware training"]
limitations:
  - Limited reasoning for complex multi-turn tasks
  - Accuracy may degrade offline for >7 days
safety_mitigations:
  - Guardrails for disallowed topics
  - PII scrubbers in data pipeline
audit_contact: compliance@qai.example
```

□ Pre-release Checklist

-
- Data validation passed (schema + bias checks)
- • Security scans passed (SBOM, dependencies)
- • Unit tests passed (tokenizer, adapters)
- • Integration tests passed (inference smoke tests)
- • NFA smoke tests passed (latency, power, safety)
- • Signed artifact present (PQC signature verified)
- • Canary deployment scheduled (24–72h telemetry)
- •

  Rollback plan documented

□ Post-release Checklist

-

- Telemetry verified for 24–72h
- • No safety incidents recorded
- • Performance within baseline thresholds
- • Drift score stable
- • Post-deploy audit completed
- •

  User feedback reviewed and logged

□ Sample Release Calendar

Window	Type	Duration	Notes
Weekly	Patch	1–2 days	Bug fixes, security patches
Monthly	Feature	3–5 days	New adapters, incremental models
Quarterly	Major	2–3 weeks	New base models, infra/tool upgrades

---

□□ Example CI/CD Job Snippet

```
jobs:
  data-validate:
    image: qai/data-validator:latest
    steps:
      - run: validate-schema --input ${DATASET}
      - run: compute-statistics --thresholds thresholds.yaml

  train-distill:
    image: qai/trainer:latest
    steps:
      - run: distill --teacher ${TEACHER} --student-config student.yaml

  package-and-sign:
```

```
    image: qai/bundler:latest
    steps:
      - run: bundle --artifact ${MODEL} --manifest modelbundle.yaml
      - run: sign --artifact bundle.tar.gz --key kms://model-signing
```

That's the **full detail of Section 14** — including model card, checklists, release calendar, and CI/CD YAML snippets.

Closing / Next steps

- The document is now bifurcated into **Architecture/Models** and **DevOps Lifecycle/Testing**.
- Next, we can create traceability matrices that link Part I requirements (taxonomy, class types, models) to Part II DevOps artifacts (tests, pipelines, release gates).

//

QAI LLM — Text Block Diagram (modules → functions → I/O → stack → DevOps/QA mapping)

Legend:
`Module Name` — short descriptor
• Functions: what it does
• I/O: typical inputs → outputs
• Tech stack: example runtimes/tools/libs (abstracted)
• QAI Model Class: target model footprints/types used here
• Data structure types: canonical data forms used/stored
• DevOps stage: lifecycle stage owning this module
• Quality standard / merit: primary NFA or compliance check it must satisfy
• Home-grown products used: specific QAI items from your stack

---

`1) Edge Interface & Device Adapters` — connectors between sensors, users, robots and LLM runtime
• Functions: input ingestion, preprocessing, sanitization, canonicalization, local caching, HW attestation

• I/O: sensor frames / IMU / images / audio / text prompts → canonical input envelopes (JSON + provenance)
• Tech stack: lightweight runtimes, embedded Python/C, tokenizers, on-device preprocessors, RTOS bindings
• QAI Model Class: Tiny (≤50M) and Small (50–500M) multimodal adapters
• Data structures: ring buffers, framed telemetry, compact feature tensors, provenance headers
• DevOps stage: Build → Unit Test → Integration Test → Package (runtime image)
• Quality standard / merit: latency p95, power draw, input validation pass-rate, safety filter success
• Home-grown products used: QAI OS device runtime, QAI OpsAgent (edge hooks), QAI Vector DB local cache

---

`2) Preprocessing & ETL / Data Ingestion` — batch & streaming ingestion pipelines
• Functions: schema validation, PII scrub, normalization, feature extraction, sampling for training/validation
• I/O: raw data (logs, DB extracts, streams) → cleaned datasets, shards, vectorizable payloads
• Tech stack: ETL jobs, Kafka/streaming, batch workers, SQL/OLAP connectors
• QAI Model Class: All classes (feeds training & RAG indices)
• Data structures: star-schema extracts, time-series tables, parquet lakes, vector embeddings
• DevOps stage: Data Validation → CI (data tests) → Publish dataset manifest
• Quality standard / merit: data integrity, provenance, bias/coverage stats, schema conformance
• Home-grown products used: QAI Datacenter ETL adapters, QAI Vector DB ingest connectors

---

`3) Vector DB & RAG Layer` — retrieval and knowledge management
• Functions: store/retrieve embeddings, hybrid index (local+global), provenance, retrieval policies, cache eviction
• I/O: query embeddings → top-K documents + provenance bundle
• Tech stack: vector store (local + centralized), ANN index, versioned indices, encryption-at-rest
• QAI Model Class: Retrieval-augmented medium/large models (adapter-enabled)
• Data structures: embedding matrices, inverted indices, signed knowledge bundles
• DevOps stage: Integration Test → Canary (retrieval correctness) → Monitoring
• Quality standard / merit: recall@k, retrieval latency, provenance completeness, signature verification
• Home-grown products used: QAI Vector DB, signed bundle manifests, QAI OpsAgent sync

---

`4) Model Layer (Core)` — base LLMs, adapters, distillation components
• Functions: base pretraining, distillation, LoRA/adapters, quantization-aware training, inference
• I/O: canonical input → logits / tokens / multimodal outputs / action envelopes
• Tech stack: transformer kernels, quantized runtimes, training orchestration (QPU hooks), model registries

• QAI Model Class: Tiny / Small / Medium / Large; modality adapters; agentic skill modules
• Data structures: model checkpoints, adapter weights, optimizer states, model cards (YAML/MD)
• DevOps stage: Experiment → Train → Evaluate → Package → Sign → Publish
• Quality standard / merit: accuracy / task success, hallucination rate, model card completeness, reproducibility
• Home-grown products used: QAI Processor compiler, QAI Datacenter trainer, QAI OS optimized kernels, model registry

---

`5) Agent Orchestration & Skills` — multi-agent planners, tool runners, safety guards
• Functions: planning, tool invocation, multi-agent coordination, policy enforcement, guardrails
• I/O: goal/intent → Plan (sequence of actions) → Execution results & observables
• Tech stack: orchestration middleware, message bus, tool adaptors (calendar, robot actuators, scripts)
• QAI Model Class: Agentic stacks using medium models + specialized executors
• Data structures: plan DAGs, action envelopes, state snapshots, audit logs
• DevOps stage: Integration → Staging Canary → Observability → Incident Runbook
• Quality standard / merit: deterministic safety properties, bounded response time, correctness of executed actions
• Home-grown products used: QAI OpsAgent (agent lifecycle), QAI OS tool adapters, signed action bundles

---

`6) Runtime & Serving` — inference hosting, autoscaling, edge orchestration
• Functions: model hosting, batching, autoscale, fallback, multi-mode serving (online/batch/offline)
• I/O: inference requests → token outputs / action envelopes / batched reports
• Tech stack: k3s/edge orchestrator, model server, cache layers, load balancer, local scheduler
• QAI Model Class: Small → Medium for edge clusters; Large for datacenter endpoints
• Data structures: runtime profiles (YAML), container images, signed bundle artifacts
• DevOps stage: Build Image → Integration Tests → Canary → Rollout → Monitoring
• Quality standard / merit: availability SLO, latency SLO, resource utilization & power envelope
• Home-grown products used: QAI OS runtime images, QAI OpsAgent orchestrator hooks, QAI Processor runtime libs

---

`7) Observability, Telemetry & Drift` — health & quality monitoring
• Functions: metrics, traces, logs, retrieval provenance capture, drift detection, user feedback ingestion
• I/O: metrics events → alerts, dashboards, retraining triggers
• Tech stack: Prometheus-style metrics, tracing, logging, drift detectors, feedback queues
• QAI Model Class: All models instrumented for quality telemetry
• Data structures: timeseries metrics, event logs, drift score tables, feedback datasets

• DevOps stage: Post-Deployment Monitoring → Trigger Retrain/Alert → Postmortem
• Quality standard / merit: SLO adherence, early drift detection, safety incident zero-tolerance
• Home-grown products used: QAI OpsAgent telemetry forwarder, QAI Datacenter retrain triggers

---

`8) Security, Compliance & PQC` — artifacts, signing, air-gap workflows
• Functions: signing, attestation, SBOM, key management, PQC signatures, RBAC & audit trails
• I/O: artifact bundles → signed manifests / attestation proofs
• Tech stack: KMS/HSM, PQC signing libs, SBOM generators, policy-as-code enforcement
• QAI Model Class: All models (signatures & manifests mandatory for long-lived deployments)
• Data structures: signed manifests, audit logs, SLSA-like build metadata
• DevOps stage: Package & Sign → Security Scans → Compliance Audit → Deploy
• Quality standard / merit: supply-chain integrity, cryptographic verification, compliance mapping (ISO/NIST)
• Home-grown products used: QAI OpsAgent signing workflows, QAI OS secure boot, QAI Datacenter key escrow

---

`9) CI/CD & Release Pipeline (Model + Infra)` — automation backbone
• Functions: data validation, training jobs, unit/integration tests, image builds, vulnerability scans, canary rollouts, rollback
• I/O: source repo + datasets → artifacts (model bundles, runtime images) → deployments
• Tech stack: GitOps/Argo, Workflow runners, artifact registry, experiment tracker, test harnesses
• QAI Model Class: applies across all releases and footprints
• Data structures: pipeline YAMLs, manifests, test reports, release notes
• DevOps stage: Entire lifecycle — orchestrates each stage above programmatically
• Quality standard / merit: gate-based release, reproducibility, test coverage thresholds, security gating
• Home-grown products used: QAI OpsAgent (CI hooks), model registry, signed bundle store

---

`10) Governance, Ethics & Model Cards` — human policy & audit layer
• Functions: policy definition, risk register, model card generation, consent & opt-out, explainability endpoints
• I/O: model metadata → human-understandable docs, audit reports, policy enforcement rules
• Tech stack: policy-as-code, compliance dashboards, automated model card generators
• QAI Model Class: all released models must have a model card & datasheet
• Data structures: model cards (YAML+MD), compliance matrices, risk logs
• DevOps stage: Design → Validation → Release → Audit

• Quality standard / merit: compliance with IEEE/NIST/NIST AI RMF mappings; fairness & transparency checks
• Home-grown products used: QAI Datacenter compliance modules, QAI OpsAgent policy enforcement

---

`11) Training & Reproducibility (QPU-assisted where applicable)` — heavy compute & experiments
• Functions: large-scale pretraining, QPU-assisted subroutines, distillation, QAT, adapter training, experiment tracking
• I/O: curated datasets → checkpoints, distilled students, adapter packages
• Tech stack: GPU/CPU clusters, QPU connectors, MLFlow-like tracker, dataset registries
• QAI Model Class: Medium→Large (teacher), Student models for edge
• Data structures: training manifests, seed/config hashes, checkpoint bundles, SBOMs
• DevOps stage: Experiment → Train → Evaluate → Archive
• Quality standard / merit: reproducibility, checkpoint integrity, training cost vs performance merit
• Home-grown products used: QAI Datacenter trainer, QAI Processor compilation pipeline, QAI Vector DB for dataset shards

---

`12) Business Integration & Org Frameworks` — mapping LLM capabilities to business processes
• Functions: map model outputs to business workflows, trigger MIS, generate reports, compliance hooks
• I/O: LLM outputs → ERP/MIS actions, tickets, dashboards
• Tech stack: connectors to ERP, workflow engines, BI tools
• QAI Model Class: office/productivity and compliance models (specialized adapters)
• Data structures: canonical action envelopes, MIS schemas, audit trails
• DevOps stage: Integration → Staging → Release → Monitoring
• Quality standard / merit: business KPI fit (e.g., MAPE for forecasts), auditability, SLAs
• Home-grown products used: Org Framework, One Business Transformation modules, QAI OpsAgent hooks

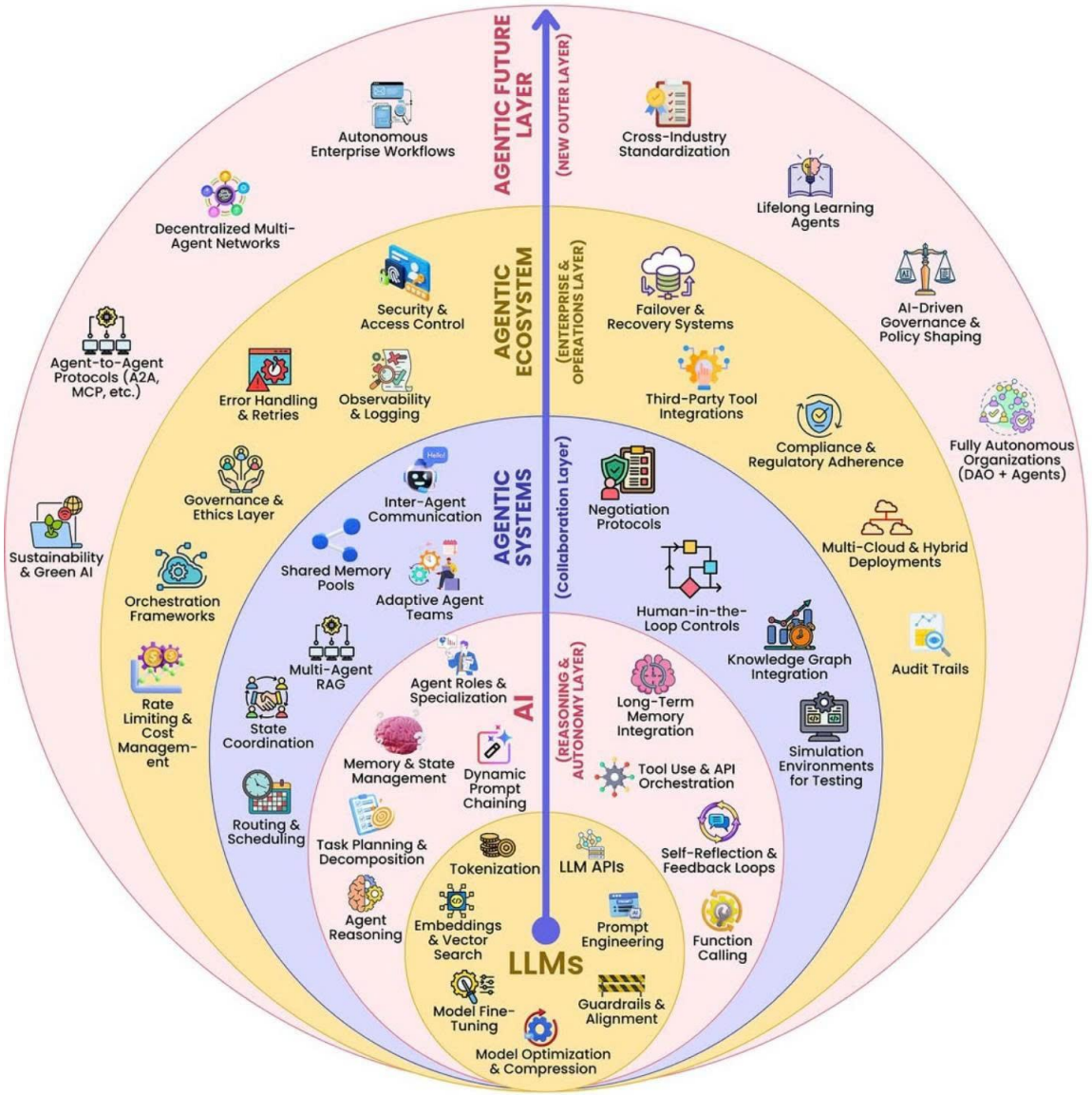
---

Final notes / how to use this diagram

• Use each module as a unit-of-deployment and attach a `Model Card + RuntimeProfile + Signed Bundle` before deployment.
• Map Requirements → Module(s) → DevOps Stage → Tests (NFA & functional) in a traceability matrix (recommended next artifact).
• For any edge/air-gapped deployment include a signed knowledge bundle + offline verification checklist.
• Prioritize building the CI/CD skeleton (section 9) early — it orchestrates all other modules and enforces gating & quality.

//

Map of Agentic AI

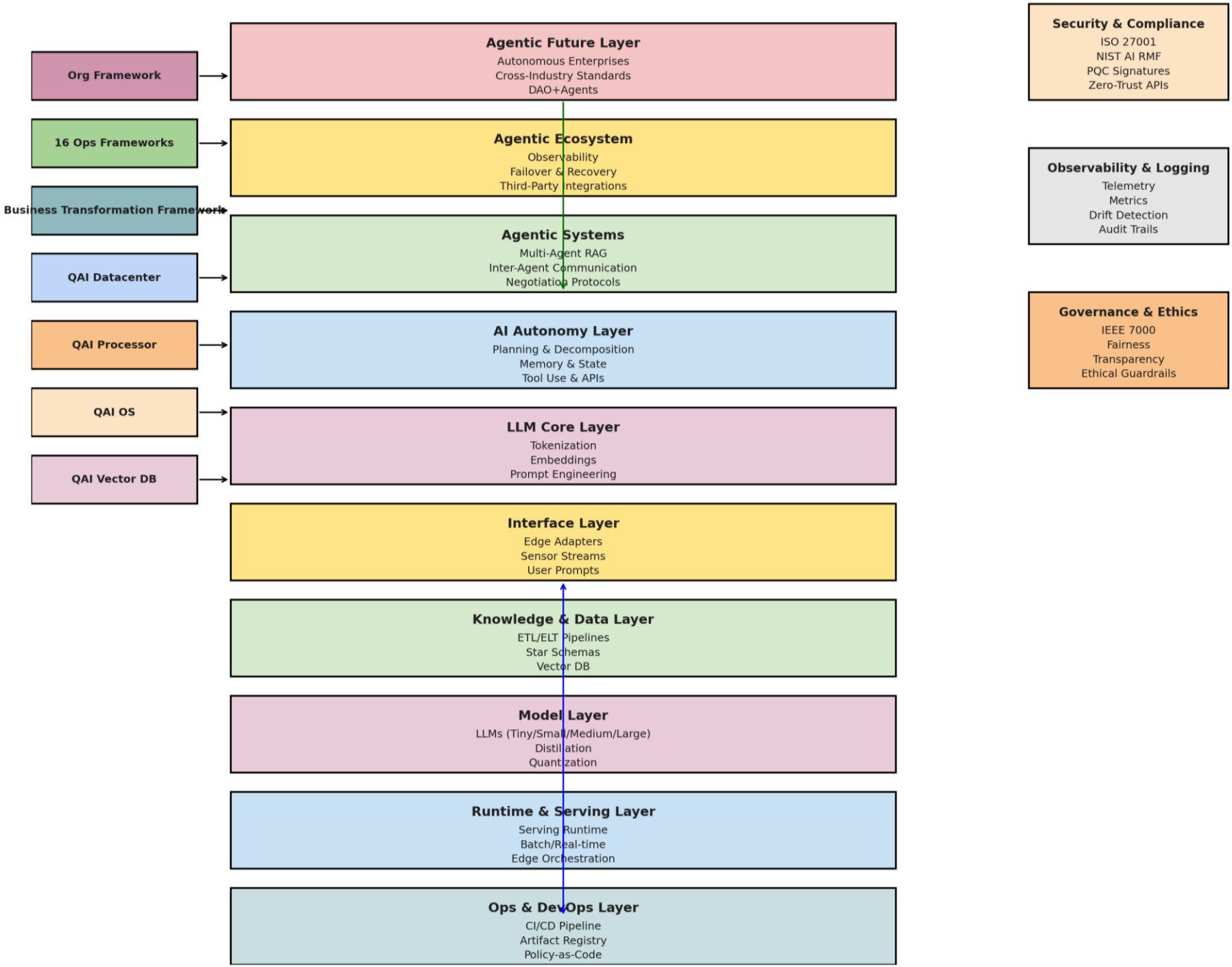


Excellent □ — Option A with your refinement makes perfect sense:

- **Core Infra & DevOps stack (Ops, Runtime, Model, Data, Interface)** will sit as the **foundation/base**.
- **Agentic progression layers (LLM Core → AI Autonomy → Agentic Systems → Ecosystem → Future)** will build upward.
- **Cross-cutting functions (Security, Observability/Logging, Governance/Ethics)** will be drawn as **vertical side bands** spanning across all layers, since they apply universally.

Final Layout Plan

1. **Bottom Foundation (QAI LLM Infra Stack)**
   - Interface Layer
   - Knowledge & Data Layer
   - Model Layer
   - Runtime & Serving Layer
   - Ops & DevOps Layer
2. **Agentic Layers (stacked above foundation)**
   - LLM Core Layer
   - AI Autonomy Layer
   - Agentic Systems
   - Agentic Ecosystem
   - Agentic Future Layer
3. **Side Cross-Cutting Functions (vertical bands)**
   - Security & Compliance
   - Observability & Logging
   - Governance & Ethics


Org Framework
16 Ops Frameworks
Business Transformation Framework
QAI Datacenter
QAI Processor
QAI OS
QAI Vector DB
Agentic Future Layer
Autonomous Enterprises
Cross-Industry Standards
DAO+Agents
Agentic Ecosystem
Observability
Failover & Recovery
Third-Party Integrations
Agentic Systems
Multi-Agent RAG
Inter-Agent Communication
Negotiation Protocols
AI Autonomy Layer
Planning & Decomposition
Memory & State
Tool Use & APIs
LLM Core Layer
Tokenization
Embeddings
Prompt Engineering
Interface Layer
Edge Adapters
Sensor Streams
User Prompts
Knowledge & Data Layer
ETL/ELT Pipelines
Star Schemas
Vector DB
Model Layer
LLMs (Tiny/Small/Medium/Large)
Distillation
Quantization
Runtime & Serving Layer
Serving Runtime
Batch/Real-time
Edge Orchestration
Ops & DevOps Layer
CI/CD Pipeline
Artifact Registry
Policy-as-Code
Security & Compliance
ISO 27001
NIST AI RMF
PQC Signatures
Zero-Trust APIs
Observability & Logging
Telemetry
Metrics
Drift Detection
Audit Trails
Governance & Ethics
IEEE 7000
Fairness
Transparency
Ethical Guardrails

Got it — thanks for the clarification. I'll produce a **text-based block diagram / flow table** that captures lifecycle steps for the **QAI_LLM_Agentic_Hybrid_With_Startup** diagram.

Each block will show:

- **Layer name**
- **Inputs → Processes → Outputs**
- **Technology stack** (classical + quantum)
- **AI model type** (Transformer, Neural Net, Hybrid, etc.)
- **Data types**
- **Lifecycle stage** (from DevOps pipeline)
- **Agility step** (Scrum/Agile relevance)
- **Startup product used** (Org Framework, 16 Ops, QAI OS, etc.)

□ QAI_LLM_Agentic_Hybrid_With_Startup — Lifecycle Flow (Text Block Diagram)

---

1. Interface Layer

- **Inputs:** Sensor streams, robotic signals, user prompts, APIs
- **Processes:** Edge adapters, preprocessing, schema normalization, validation
- **Outputs:** Canonical input envelopes, preprocessed feature vectors
- **Technology:** RTOS bindings, lightweight runtimes, Python/C preprocessors
- **AI Models:** QAI Tiny Transformer (≤50M), QAI Edge Neural Network
- **Data Types:** JSON envelopes, sensor frames, telemetry packets
- **Lifecycle Stage:** Requirements → Data Engineering
- **Agility Step:** Backlog user stories, quick sprint demos of adapters
- **Startup Products:** QAI OS (edge runtime), QAI OpsAgent (adapter hooks)

---

2. Knowledge & Data Layer

- **Inputs:** Raw + preprocessed data, DB extracts, external feeds
- **Processes:** ETL/ELT pipelines, schema validation, time-series indexing, embedding generation
- **Outputs:** Clean datasets, vector embeddings, manifests, provenance logs
- **Technology:** QAI Vector DB, RAG hybrid retrievers, Star schema warehousing
- **AI Models:** Embedding models (QAI BERT-lite, QAI Transformer encoders)
- **Data Types:** Parquet, vector arrays, SQL tables, provenance metadata
- **Lifecycle Stage:** Data Engineering → Data Validation
- **Agility Step:** Incremental ingestion in sprints, backlog grooming for new feeds
- **Startup Products:** QAI Datacenter ETL, QAI Vector DB, QAI Data Processors

---

3. Model Layer

- **Inputs:** Datasets + embeddings + configs
- **Processes:** Pretraining, distillation, quantization, LoRA/adapters, hybrid QPU-assisted training
- **Outputs:** Model checkpoints, adapter weights, compressed student models
- **Technology:** QAI Processor (quantum-classical hybrid cores), GPU/NPU, quantum ansatz solvers
- **AI Models:** QAI Transformers, QAI Hybrid Neural Networks, QAI Spiking NNs (for CPS)
- **Data Types:** Checkpoints, YAML configs, tensor weights
- **Lifecycle Stage:** Design → Training → Testing
- **Agility Step:** Iterative experiments, client feedback on prototypes
- **Startup Products:** QAI Processor, QAI OS kernels, QAI Datacenter trainer

---

4. Runtime & Serving Layer

- **Inputs:** Signed model bundles, runtime profiles
- **Processes:** Model hosting, inference serving, autoscaling, fallback execution
- **Outputs:** Predictions, decisions, action envelopes
- **Technology:** QAI OS runtimes, k3s/edge orchestrator, model server, load balancer
- **AI Models:** Medium-size QAI Transformers (cloud), Tiny Edge Models (device)
- **Data Types:** Inference requests/responses, action JSONs, telemetry
- **Lifecycle Stage:** Deployment → Monitoring
- **Agility Step:** Continuous delivery cycles, feedback-driven release planning

- **Startup Products:** QAI OS runtime images, QAI OpsAgent orchestrator

---

5. Ops & DevOps Layer

- **Inputs:** Source code, configs, datasets, tests
- **Processes:** CI/CD pipelines, data validation jobs, build → test → sign → deploy
- **Outputs:** Signed model artifacts, release notes, audit logs
- **Technology:** GitOps, MLFlow tracker, artifact registry, PQC signatures
- **AI Models:** Meta models for automation (AutoML/QAI OpsAgent crew)
- **Data Types:** Pipeline YAMLs, SBOMs, manifests, reports
- **Lifecycle Stage:** CI/CD → Release → Monitoring
- **Agility Step:** Sprint-end delivery, retrospective integration
- **Startup Products:** QAI Ops Frameworks (16 Ops), Org Framework (traceability)

---

6. LLM Core Layer

- **Inputs:** Canonical data, embeddings, prompts
- **Processes:** Tokenization, embeddings, prompt processing, QPU offloading for optimization
- **Outputs:** Predictions, contextual embeddings, token streams
- **Technology:** QAI Transformer cores, Hybrid Quantum Circuits, Function calling APIs
- **AI Models:** QAI Transformers (Tiny → Large), QAI RNNs for sequence tasks
- **Data Types:** Tokens, embeddings, logits, structured output envelopes
- **Lifecycle Stage:** Inference → Validation → Monitoring
- **Agility Step:** Feature delivery sprints, iterative model cards
- **Startup Products:** QAI Processor, QAI OS, QAI Datacenter

---

7. AI Autonomy Layer

- **Inputs:** Goals, prompts, context windows
- **Processes:** Task planning, memory/state management, tool use, self-reflection
- **Outputs:** Plans, decomposed tasks, execution traces
- **Technology:** Multi-agent orchestration, policy engines, memory stores

- **AI Models:** QAI Agent Models, Hybrid Planning NNs, Reinforcement Learning QAI agents
- **Data Types:** DAGs (plans), state snapshots, logs
- **Lifecycle Stage:** Testing → Deployment → Monitoring
- **Agility Step:** Sprint-based planning updates, backlog for new agent skills
- **Startup Products:** QAI OpsAgent, QAI Org Framework

---

8. Agentic Systems Layer

- **Inputs:** Plans, tasks, inter-agent messages
- **Processes:** Multi-agent RAG, negotiation protocols, governance enforcement
- **Outputs:** Coordinated actions, updated policies, inter-agent logs
- **Technology:** Agentic frameworks, swarm intelligence, quantum-enhanced optimization
- **AI Models:** QAI Multi-Agent NN, QAI Swarm RL Models
- **Data Types:** Messages, policies, audit logs
- **Lifecycle Stage:** Deployment → Monitoring → Incident Management
- **Agility Step:** Sprint-end demos of multi-agent use cases
- **Startup Products:** QAI Org Framework, Business Transformation Framework

---

9. Agentic Ecosystem Layer

- **Inputs:** Ecosystem events, external integrations
- **Processes:** Compliance enforcement, observability, failover, cross-cloud orchestration
- **Outputs:** Audit reports, dashboards, compliance certificates
- **Technology:** Multi-cloud orchestration, compliance dashboards, observability stacks
- **AI Models:** Governance NNs, Risk Monitoring Agents
- **Data Types:** Compliance matrices, telemetry streams, SLA dashboards
- **Lifecycle Stage:** Governance → Compliance Audits
- **Agility Step:** Incremental compliance sprints, stakeholder reviews
- **Startup Products:** Business Transformation Framework, 16 Ops Frameworks

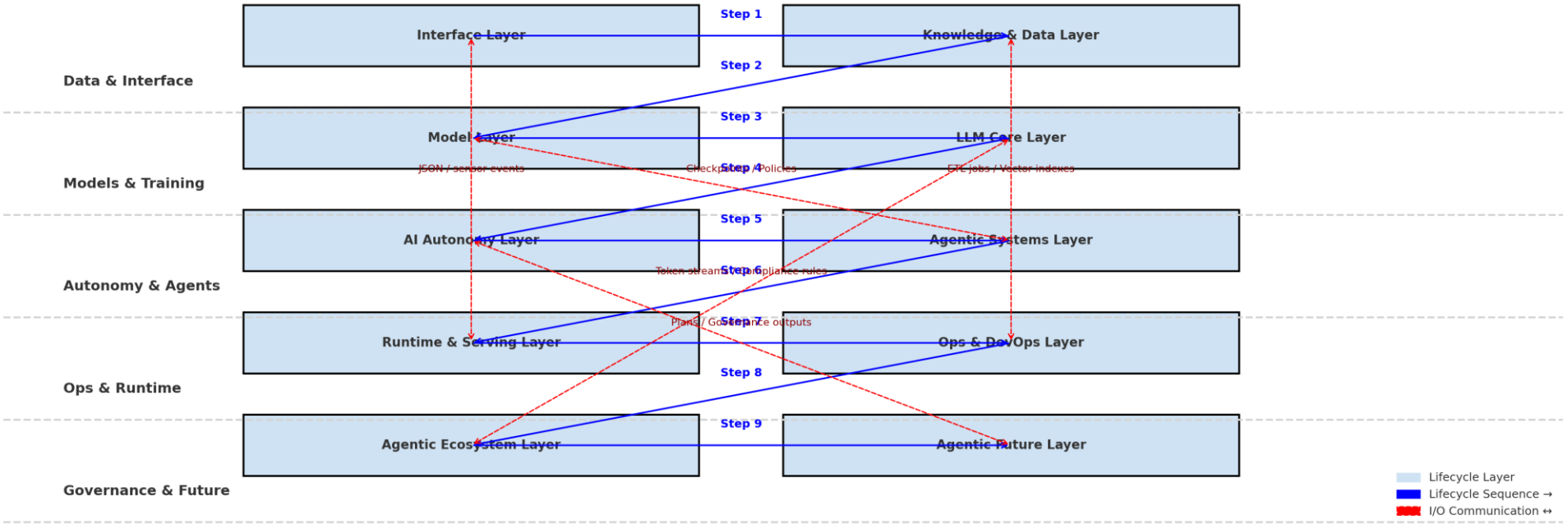
---

10. Agentic Future Layer

- **Inputs:** Strategic goals, long-term data, sustainability metrics
- **Processes:** Autonomous enterprise orchestration, DAO integration, societal-level optimization
- **Outputs:** Policy recommendations, autonomous organization outputs, innovation logs
- **Technology:** Blockchain + PQC, DAO frameworks, AI-Governance platforms
- **AI Models:** QAI DAO Agents, QAI Policy LLMs, Societal-scale RL agents
- **Data Types:** Policies, governance tokens, sustainability KPIs
- **Lifecycle Stage:** Strategic Planning → Innovation → Evolution
- **Agility Step:** Quarterly innovation sprints, roadmap reviews
- **Startup Products:** Business Transformation Framework, QAI CleanEarth (future extension)

 This gives you a**complete text block diagram flow** for the lifecycle across *all layers* of the hybrid framework, tying in your startup tools and Agile cycle.

//



For more details contact: vijaymohire@gmail.com