ReGraph Whitepaper

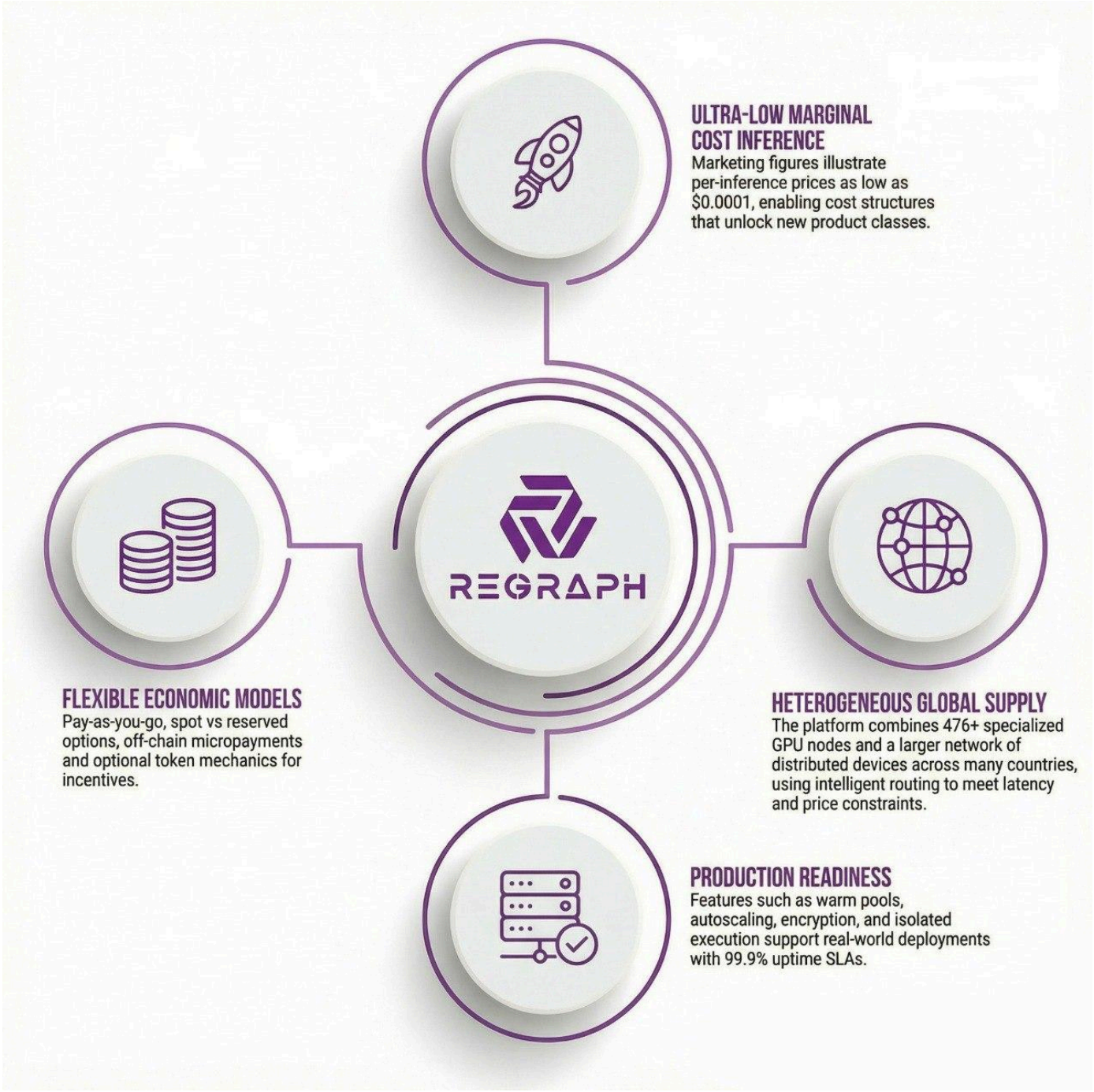
Version 1.01 — January 2026

Authors: Ilya Dushin, Gabriel Mikhaeli, ReGraph Core Team

Disclaimer: This document presents the conceptual, architectural, economic, and operational design of ReGraph — a decentralized AI compute marketplace. It is intended to explain the rationale and mechanics behind the project in detail. Implementation specifics, low-level API references, and product marketing materials are out of scope.

1. Executive Summary

ReGraph (https://regraph.tech) is a decentralized AI compute marketplace that aggregates heterogeneous compute resources — GPUs, TPUs, NPUs, and even smartphone processors — into a global, production-grade platform for AI inference and training.



It is designed to deliver dramatically lower unit costs (up to 80% cheaper vs. incumbent cloud providers), fine-grained pay-as-you-go billing, and novel monetization opportunities for hardware owners. At the same time, ReGraph emphasizes enterprise-grade security (end-to-end encryption, SOC2 readiness, isolated execution), predictable SLAs, and developer ergonomics.

This whitepaper presents the motivation, architecture, economic model, security posture, operational strategies, governance considerations, and roadmap for ReGraph. The document grounds each design decision in the problems it solves and explains why those choices matter for both developers and providers.

Key highlights

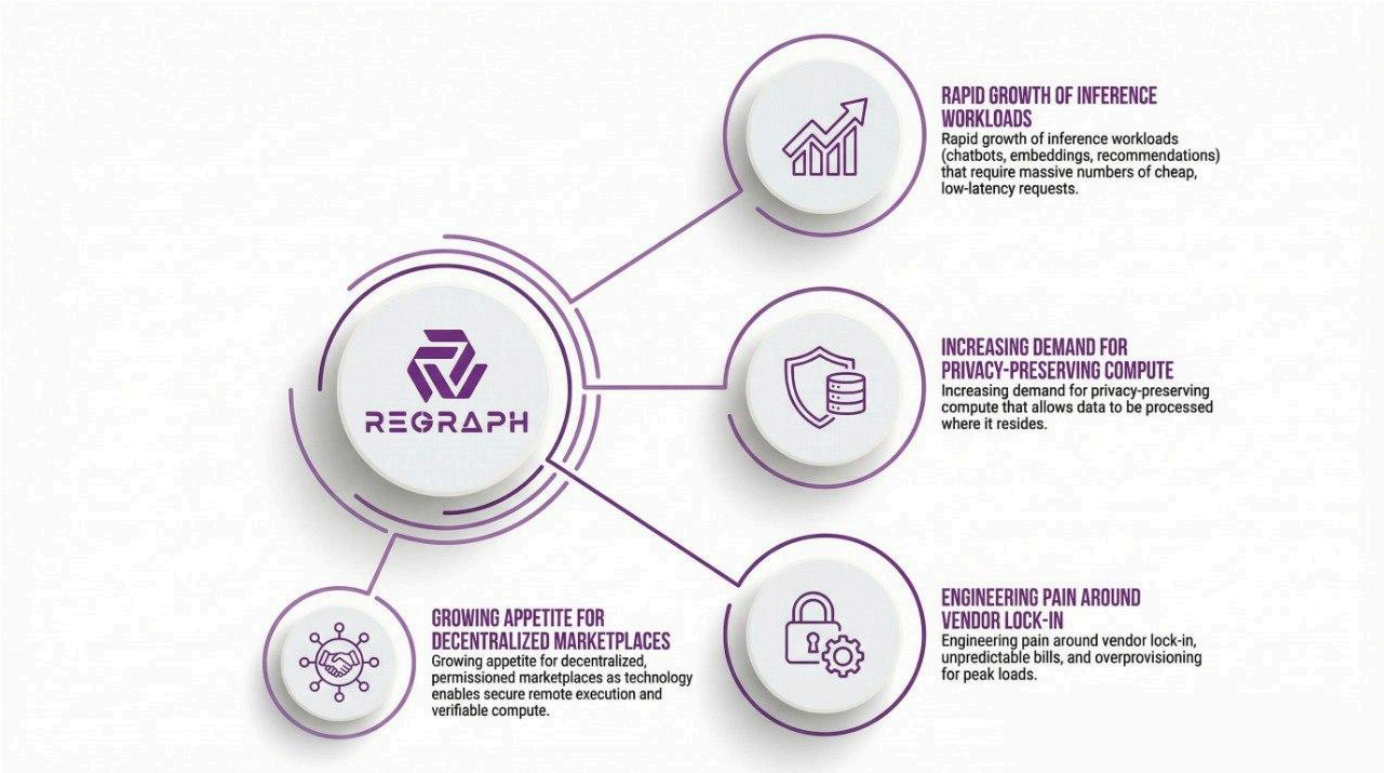
- Ultra-low marginal cost inference: marketing figures illustrate per-inference prices as low as $0.0001, enabling cost structures that unlock new product classes.
- Heterogeneous global supply: the platform combines 476+ specialized GPU nodes and a larger network of distributed devices across many countries, using intelligent routing to meet latency and price constraints.
- Production readiness: features such as warm pools, autoscaling, encryption, and isolated execution support real-world deployments with 99.9% uptime SLAs.
- Flexible economic models: pay-as-you-go, spot vs reserved options, off-chain micropayments and optional token mechanics for incentives.

2. Context and Market Dynamics

The emergence of large language models (LLMs), diffusion-based generative models, and other compute-intensive AI architectures has dramatically increased demand for GPU/accelerator compute. Traditional cloud providers supply this compute but at high unit prices and with inflexible commercial models (e.g., minimum commitments, reserved instances, opaque spot markets). Meanwhile, across global datacenters, colocation facilities, enterprises, and even everyday devices, there exists a large pool of underutilized compute that could be monetized.

Market signals

- Rapid growth of inference workloads (chatbots, embeddings, recommendations) that require massive numbers of cheap, low-latency requests.
- Increasing demand for privacy-preserving compute that allows data to be processed where it resides.
- Engineering pain around vendor lock-in, unpredictable bills, and overprovisioning for peak loads.
- Growing appetite for decentralized, permissioned marketplaces as technology enables secure remote execution and verifiable compute.



These dynamics create an opening for a marketplace that aligns supply and demand, reduces unit economics, and introduces new degrees of freedom: choose nodes by price, latency, hardware type, and privacy characteristics.

3. Problems and Opportunity Space

The rapid proliferation of machine learning, and particularly large-scale generative models, has transformed how software is built, deployed, and monetized. What was once experimental research infrastructure has become core production plumbing for services ranging from conversational agents to personalization engines, recommendation systems, and automated content generation. Yet, while the demand for accelerator-backed compute has exploded, the supply ecosystem and consumption patterns remain suboptimal: compute is expensive, concentrated, inflexible, and often poorly matched to the diverse latency, cost, privacy, and regulatory constraints of real-world applications.

This section frames the core problems that create frictions across the AI stack and identifies the opportunity space where a decentralized, marketplace-driven approach can provide outsized value. Instead of cataloguing only technical shortcomings (e.g., slow cold starts or memory limits), the introduction below situates those technical pain points within broader economic, operational, and regulatory contexts. The goal is to make explicit why these problems matter to different stakeholders — developers, enterprises, hardware owners, researchers, and regulators — and to quantify (where possible) the size and shape of the opportunity that emerges when those frictions are resolved.

Why problem framing matters:

- **Strategic clarity:** Precise problem articulation exposes which constraints are technical (e.g., memory footprints), economic (e.g., unit costs and minimum commitments), social (e.g., access to compute for research or startups), and regulatory (e.g., data residency and privacy). Solutions that only address a subset of these constraints fail to win broad adoption.
- **Design constraints:** Every architectural and economic decision in a marketplace trades off cost, latency, reliability, and trust. A carefully reasoned problem statement gives rise to measurable design goals and prioritization criteria used across the rest of the whitepaper.
- **Stakeholder alignment:** Different parties experience problems differently. Developers want predictable, cheap inference. Enterprises want compliance and SLAs. Hardware owners seek clear monetization and low operational burden. Research programs

require bursty, affordable access. A shared problem taxonomy enables aligned incentive design.

3.1. High Cost and Opaque Pricing

Problem: Cloud GPU and managed inference services are expensive per GPU-hour and per-inference, with additional hidden costs (data egress, storage, reserved commitments).
Why it matters: High unit costs prevent smaller developers and startups from offering high-volume AI services profitably; they force large players to overcommit capital or accept inferior latency.

3.2. Vendor Lock-in & Limited Portability

Problem: Proprietary services and optimized toolchains create migration friction.
Why it matters: Organizations want to avoid single-provider dependency for pricing resilience and governance; portability fosters competition and innovation.

3.3. Underutilized Global Compute

Problem: Hardware is underutilized in research labs, enterprise clusters, university HPC systems, and end-user devices.
Why it matters: Monetizing this idle compute improves resource efficiency and lowers costs for consumers while offering income for providers.

3.4. Data Privacy, Residency, and Compliance

Problem: Many AI workloads handle sensitive data and cannot be processed in public cloud regions or third-party systems without explicit controls.
Why it matters: Regulated industries require guarantees about where and how data is processed; offering on-prem/private pool options unlocks high-value enterprise customers.

3.5. Reliability at Scale

Problem: Decentralized resources are heterogeneous and unreliable by nature, complicating production SLAs.
Why it matters: Enterprise adoption hinges on predictable performance and reliability; the marketplace must architect around uncertainty.

Each problem defines a design constraint and informs the platform's technical and economic stack.

4. Vision, Mission, and Design Goals

Vision: A neutral, secure, and efficient global marketplace where compute supply meets demand dynamically, enabling affordable, privacy-respecting, and scalable AI for everyone.

Mission: To unlock underutilized compute and align economic incentives to deliver low-cost inference and training without compromising security, compliance, or developer productivity.

Design goals:

- Cost-efficiency: Reduce per-inference and per-GPU-hour costs through competition, spot markets, and efficient matching.

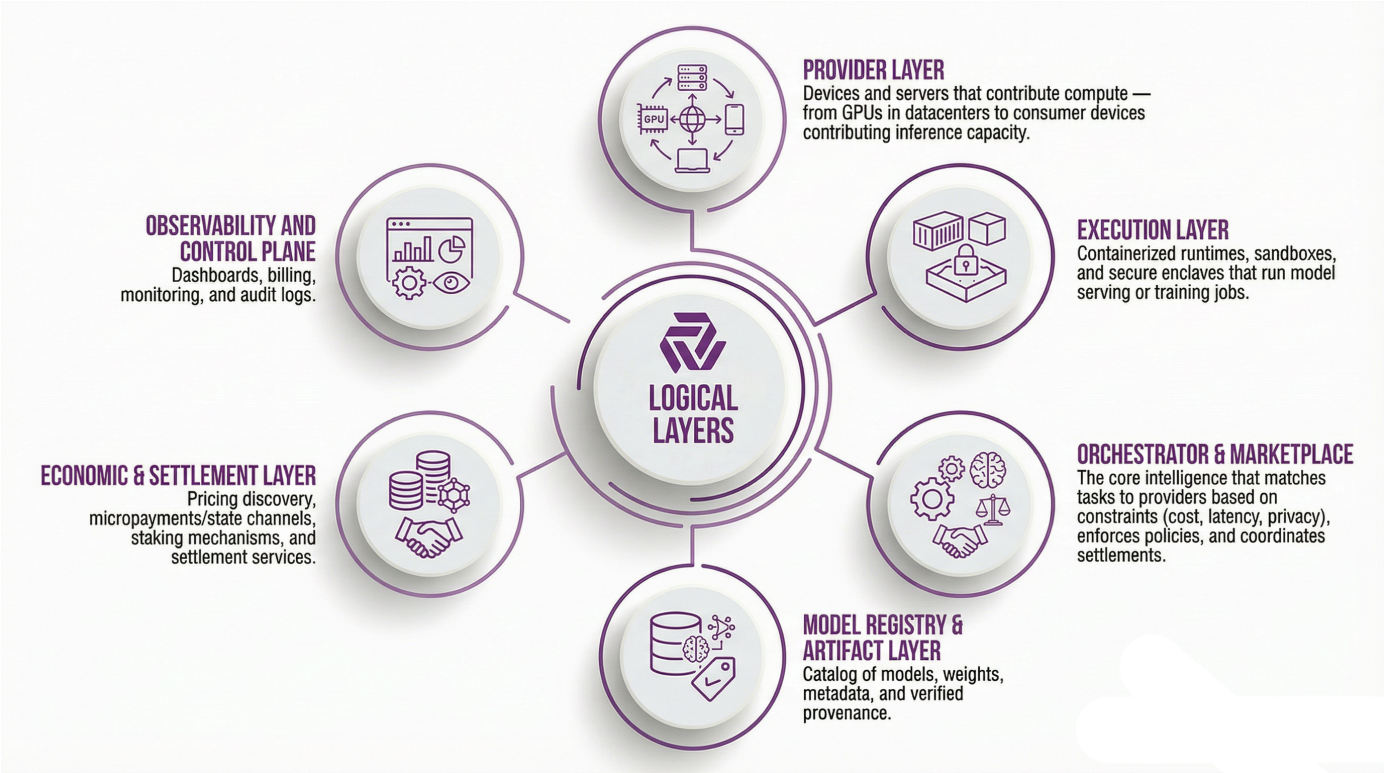
- Reliability and predictability: Support production SLAs and enterprise grade features.
- Privacy and control: Guarantee that customers can enforce data residency and execution policies.
- Provider fairness: Ensure transparent, timely compensation and mechanisms to prevent gaming.
- Developer ergonomics: Offer familiar, model-agnostic interfaces and tooling to accelerate adoption.
- Incremental decentralization: Start with pragmatic centralized orchestrator elements for operational control, with a roadmap toward decentralized components and governance.

Why these goals matter: Balancing cost, trust, and operational feasibility is critical. Purely decentralized systems risk safety and enterprise adoption; fully centralized approaches lose the benefits of distributed supply. ReGraph adopts a pragmatic hybrid pathway.

5. High-Level System Overview

At a high level, ReGraph composes the following logical layers:

- Provider Layer: Devices and servers that contribute compute — from GPUs in datacenters to consumer devices contributing inference capacity.
- Execution Layer: Containerized runtimes, sandboxes, and secure enclaves that run model serving or training jobs.
- Orchestrator & Marketplace: The core intelligence that matches tasks to providers based on constraints (cost, latency, privacy), enforces policies, and coordinates settlements.
- Model Registry & Artifact Layer: Catalog of models, weights, metadata, and verified provenance.
- Economic & Settlement Layer: Pricing discovery, micropayments/state channels, staking mechanisms, and settlement services.
- Observability and Control Plane: Dashboards, billing, monitoring, and audit logs.



The platform intentionally separates concerns: orchestration and market logic are decoupled from execution environments, enabling private pools or on-prem controllers to operate under local policies while optionally leveraging the broader marketplace.

6. Architectural Components and Rationale

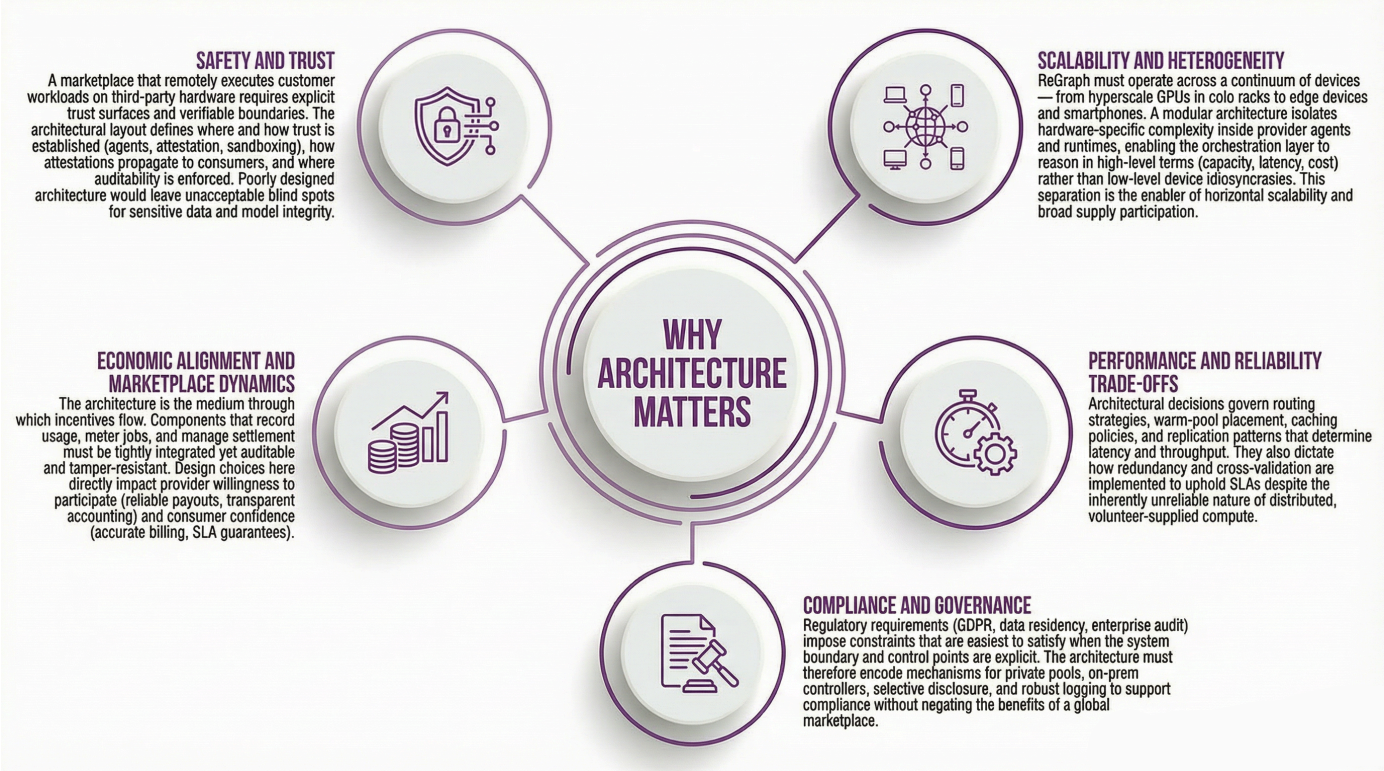
The architecture of ReGraph is the blueprint that turns a conceptual marketplace for AI compute into a resilient, secure, and economically viable system. At its core, this architecture must reconcile contradictory pressures: extreme cost-efficiency versus predictable enterprise-grade reliability; decentralized, heterogeneous supply versus centralized policy and compliance; rapid developer ergonomics versus rigorous security and verifiability. The components and their interactions are not incidental implementation details — they are the instruments by which the platform guarantees trust, performance, and fair economics.

Why architecture matters?

- Safety and trust: A marketplace that remotely executes customer workloads on third-party hardware requires explicit trust surfaces and verifiable boundaries. The architectural layout defines where and how trust is established (agents, attestation, sandboxing), how attestations propagate to consumers, and where auditability is enforced. Poorly designed architecture would leave unacceptable blind spots for sensitive data and model integrity.
- Scalability and heterogeneity: ReGraph must operate across a continuum of devices — from hyperscale GPUs in colo racks to edge devices and smartphones. A modular architecture isolates hardware-specific complexity inside provider agents and runtimes, enabling the orchestration layer to reason in high-level terms (capacity, latency, cost) rather than low-level device idiosyncrasies. This separation is the enabler of horizontal scalability and broad supply participation.
- Economic alignment and marketplace dynamics: The architecture is the medium through which incentives flow. Components that record usage, meter jobs, and manage settlement must be tightly integrated yet auditable and tamper-resistant. Design choices here directly impact provider willingness to participate (reliable payouts, transparent accounting) and consumer confidence (accurate billing, SLA guarantees).
- Performance and reliability trade-offs: Architectural decisions govern routing strategies, warm-pool placement, caching policies, and replication patterns that determine latency and throughput. They also dictate how redundancy and cross-validation are

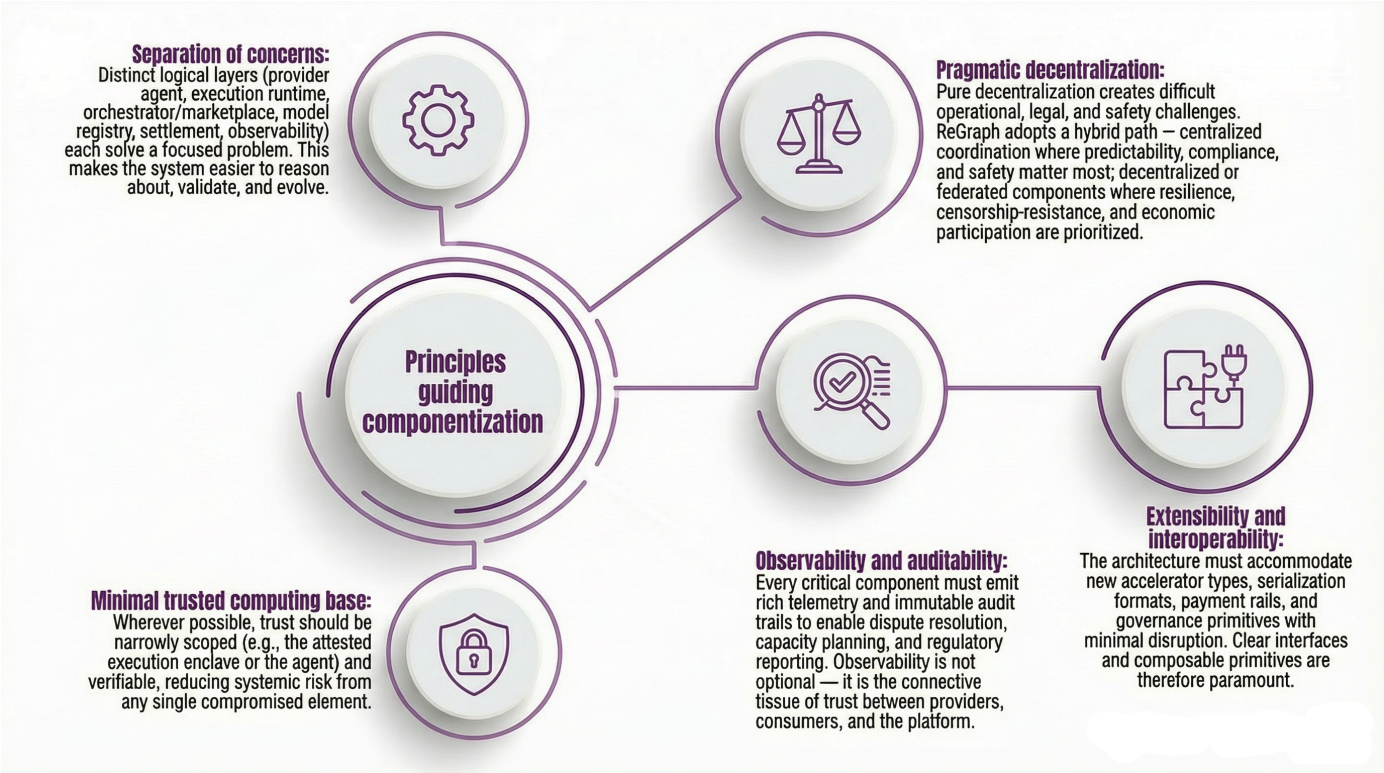
implemented to uphold SLAs despite the inherently unreliable nature of distributed, volunteer-supplied compute.

- Compliance and governance: Regulatory requirements (GDPR, data residency, enterprise audit) impose constraints that are easiest to satisfy when the system boundary and control points are explicit. The architecture must therefore encode mechanisms for private pools, on-prem controllers, selective disclosure, and robust logging to support compliance without negating the benefits of a global marketplace.



Principles guiding componentization

- Separation of concerns: Distinct logical layers (provider agent, execution runtime, orchestrator/marketplace, model registry, settlement, observability) each solve a focused problem. This makes the system easier to reason about, validate, and evolve.
- Minimal trusted computing base: Wherever possible, trust should be narrowly scoped (e.g., the attested execution enclave or the agent) and verifiable, reducing systemic risk from any single compromised element.
- Extensibility and interoperability: The architecture must accommodate new accelerator types, serialization formats, payment rails, and governance primitives with minimal disruption. Clear interfaces and composable primitives are therefore paramount.
- Pragmatic decentralization: Pure decentralization creates difficult operational, legal, and safety challenges. ReGraph adopts a hybrid path — centralized coordination where predictability, compliance, and safety matter most; decentralized or federated components where resilience, censorship-resistance, and economic participation are prioritized.
- Observability and auditability: Every critical component must emit rich telemetry and immutable audit trails to enable dispute resolution, capacity planning, and regulatory reporting. Observability is not optional — it is the connective tissue of trust between providers, consumers, and the platform.



6.1. Provider Agent & Edge Nodes

What: A lightweight, cross-platform agent (Linux, Windows, macOS, Android) that interfaces a provider's hardware with the marketplace. It reports capabilities, receives tasks, enforces isolation, and streams telemetry.

Why: A standardized agent enables discovery of heterogeneous hardware, enforces execution contracts, collects usage for billing, and provides a trust surface for attestation. Without a uniform agent, building a consistent marketplace is infeasible.

Key features:

- Hardware discovery and capability reporting (GPU memory, compute capability, accelerator types).
- Secure communication channels with the Orchestrator.
- Local policy controls (availability windows, reserved compute).
- Execution supervision (container lifecycle, resource cgroups).
- Optional enclave support and attestation integration.

Trade-offs and rationale:

- Agent footprint must be minimal to enable smartphone participation.
- Cross-platform complexity is necessary to unlock broad supply but increases engineering cost.
- Optional attestation allows flexible adoption: providers can join without TEEs, while privacy-sensitive consumers require attested nodes.

6.2. Orchestrator & Marketplace Layer

What: The marketplace brain responsible for matching, scheduling, reputation accounting, price discovery, and policy enforcement. It exposes developer-facing flows and provider onboarding.

Why: Markets require centralized (or coordinatored) intelligence to balance supply and demand efficiently. The Orchestrator enforces safety policies, aggregates supply, and coordinates settlements. A fully decentralized rendezvous layer is a future goal but initial centralization is pragmatic for safety, compliance, and predictable SLAs.

Core responsibilities:

- Matching engine for tasks → nodes.
- Pricing discovery and negotiation.
- Reputation tracking and QoS monitoring.
- Policy enforcement (model licensing, content restrictions).
- Aggregation for micropayment batching and settlement.

Design principles:

- Deterministic, auditable matching logic to reduce dispute surface.
- Extensible policy framework (e.g., for data residency).
- SLO/SLA awareness during scheduling.

6.3. Model Registry and Provenance

What: A secure registry storing model artifacts (weights, tokenizer, config), metadata (licensing, provenance, allowed usage), and verified hashes/signatures.

Why: Model provenance and licensing are paramount: running models with ambiguous provenance incurs legal and safety risks. The Registry ensures integrity, facilitates versioning, and enables trusted deployments across distributed nodes.

Key aspects:

- Signed artifacts and content-addressable storage for reproducibility.
- Metadata about resource requirements, recommended runtime flags (e.g., quantization), and permitted geographies.
- Access-control lists for private models.

6.4. Settlement & Economic Layer

What: The financial backbone enabling micropayments, provider earnings, platform fees, and optional token mechanics.

Why: Per-inference pricing at sub-cent levels requires efficient financial flows. Conventional on-chain settlement per transaction is cost-prohibitive. The layer supports off-chain aggregation, payment channels, and periodic settlements or fiat consolidation.

Components:

- Price discovery marketplaces and spot vs reserved markets.
- Off-chain payment channels or state channels for high-frequency micropayments.
- Periodic batching and settlement in fiat or crypto.
- Accounting systems for invoicing, taxation, and compliance.

6.5. Control Plane, Observability, and Management

What: Dashboards and systems providing real-time visibility into usage, costs, performance, provider health, and compliance.

Why: Developers and enterprises must manage costs, detect anomalies, and enforce governance. Providers need transparent earnings reports. Observability supports trust and operational troubleshooting.

Features:

- Metrics: latency histograms, per-model throughput, provider uptime.
- Alerts and usage budgets.
- Per-job audit trails for compliance.
- Billing feeds and forecasting.

7. Scheduling, Routing, and Performance Strategies

The Scheduling, Routing, and Performance Strategies section addresses the core runtime decisions that determine how well ReGraph translates raw, heterogeneous supply into predictable, cost-effective, and secure AI compute for consumers. In a decentralized marketplace, the value of aggregated compute is not simply the sum of available GPUs or accelerators; it is a function of how intelligently the system places work onto nodes, moves or caches model artifacts, and coordinates execution under constraints such as latency, cost, privacy, and reliability. This section explains why those operational choices matter, defines the key trade-offs, and frames the principles that guide the design of ReGraph's matchmaking and execution subsystems.

Key constraints and trade-offs

- Latency vs Cost: Prioritizing the cheapest possible nodes will often increase network latency and cold-start likelihood. Conversely, routing to the closest warm node minimizes response time but may charge a premium. The scheduler must optimize along a multi-dimensional Pareto frontier aligned with consumer preferences.
- Throughput vs Determinism: Combining requests into batches or aggregating similar workloads improves GPU utilization and reduces per-request cost, but introduces queuing and variable latency. Different workloads demand different points on this spectrum.
- Consistency vs Availability: Redundant execution and cross-validation increase correctness assurances but at the expense of additional resource consumption and cost. The system must apply redundancy selectively based on risk, value, and SLA levels.
- Flexibility vs Predictability: Spot-style dispatch enables very low prices but introduces preemption risk. Reserved or private capacity provides predictability at higher cost. Scheduling must support mixed-mode strategies to satisfy diverse customer requirements.

Principles that guide ReGraph's scheduling design

- Constraint-aware matching: Every scheduling decision is driven by explicit constraints — maximum acceptable latency, required hardware features, privacy restrictions, and cost ceilings. These constraints must be first-class inputs into routing algorithms.
- Multi-objective optimization: Scheduling is intrinsically multi-objective. ReGraph uses weighted optimization heuristics (and later, policy-driven or learned strategies) to balance latency, cost, reliability, and provider fairness rather than optimizing a single metric.
- Predictive and adaptive behavior: The system leverages telemetry, historical patterns, and predictive models to pre-warm containers, prefetch model shards, and adjust routing proactively to anticipated demand spikes.
- Hierarchical decision-making: To scale, scheduling decisions occur at multiple levels — global market-level price discovery, regional edge-level placement, and local node-level execution. Each level applies appropriate policy and telemetry to refine choices.
- Transparency and auditability: Scheduling outcomes influence economic settlements and provider reputation. Decisions and their rationales must be auditable to support disputes, performance reviews, and continual improvement.

- Policy encapsulation and extensibility: Scheduling must respect business and regulatory policies delivered as pluggable constraints — for example, tenant-level data residency restrictions or enterprise whitelists. This enables flexible adoption across verticals and geographies.

Measurable objectives and success criteria

- Tail latency percentiles (P50/P95/P99) for interactive workloads and target thresholds (e.g., P95 < 100ms for “low-latency” class).
- Cost per inference and cost per GPU-hour compared to baseline cloud alternatives.
- Cold-start incidence and average cold-start duration for popular models.
- Match rate and scheduling efficiency: fraction of tasks served by first-choice node vs. reroutes.
- Provider utilization and fairness metrics: how workloads and earnings are distributed across providers and how reputation maps to task allocation.
- SLA adherence: uptimes and fulfillment rates for reserved capacity customers.

7.1. Latency-First Routing

Objective: Serve interactive inference requests under strict latency targets (e.g., sub-100ms for some use cases).

Approach:

- Edge-first routing prioritizes nodes near the request source.
- Warm pools of preloaded model containers on strategically located nodes.
- Proactive caching of model weights and quantized variants.
- Pre-warming model replicas in regions with high demand.

Why: Many applications (chatbots, AR/VR, real-time personalization) are latency-sensitive; excessive latency degrades UX. Geographic proximity and warm containers materially reduce round-trip times and cold-start latencies.

7.2. Cost-First Routing & Spot Markets

Objective: Minimize cost for non-latency-sensitive workloads (bulk inference, batch processing, non-realtime training).

Approach:

- Spot-style dispatch to the lowest-price nodes that meet minimum resource constraints.
- Deferred execution windows and preemption hooks.
- Batch aggregation to amortize overhead across many inputs.

Why: Not all workloads require sub-100ms responses; leveraging cheaper providers reduces costs, enabling previously unprofitable services (e.g., ultra-cheap embedding calculations for large corpora).

7.3. Warm Pools, Caching, and Cold-Start Mitigation

Problem: Model weight transfer and runtime initialization cause cold-start penalties, sometimes minutes for large LLMs.

Solutions:

- Maintain warm pools (hot containers with artifacts loaded) for popular models in core regions.

- Predictive pre-warming based on usage patterns and warm-up costs.
- Hybrid strategies: small local proxies for quick partial responses combined with backend completions.

Tradeoffs:

- Warm pools consume resources and entail standing costs; revenue premium or reserved capacity options can cover costs.
- Intelligent sizing and eviction policies are essential for cost control.

7.4. Sharding, Model Parallelism, and Mixed-Precision Techniques

Large models may not fit a single node's memory. ReGraph supports:

- Model sharding across nodes, using layers of model-parallel frameworks.
- Quantized model execution (8-bit, 4-bit) and kernel-level optimizations (TensorRT, ONNX Runtime).
- Adaptive precision: trade accuracy for memory and latency depending on SLA.

Rationale: Supporting a broad range of model sizes increases marketplace usability and broadens supply options. Mixed-precision reduces resource needs and increases fit rate on smaller nodes.

7.5. Batch Aggregation and Throughput Optimization

For throughput-oriented jobs (embedding generation, large corpus inference):

- Aggregate similar requests to enable vectorized execution.
- Use data pipelines and asynchronous scheduling to maximize GPU utilization.
- Provide backpressure and queueing semantics to avoid overloading nodes.

Result: Superior cost-efficiency for high-volume batch operations.

8. Security, Privacy, and Trust

A robust security model is central to marketplace adoption, especially for enterprise and regulated workloads.

8.1. Threat Model and Objectives

Threats considered:

- Malicious providers altering inputs/outputs or exfiltrating sensitive data.
- Compromised provider agents executing arbitrary code.
- Man-in-the-middle or replay attacks on control and data channels.
- Sybil attacks and reputation manipulation.

Security objectives:

- Maintain confidentiality and integrity of data and model artifacts.
- Provide verifiable execution guarantees and provenance.
- Prevent fraud and ensure correct financial settlement.

8.2. Confidentiality: Encryption & Data Minimization

Mechanisms:

- TLS for control and data channels; optional end-to-end encryption for payloads where only the client holds the decryption key.
- Data minimization strategies: avoid persistent storage of sensitive artifacts; strict retention policies and ephemeral execution.
- Client-side encryption with secure key-sharing for selected nodes (e.g., using secure enclaves for decryption).

Why: Enterprises require assurances that sensitive data (PHI, PII) is not exposed. Providing encryption that minimizes third-party visibility is necessary for regulatory compliance.

8.3. Integrity: Attestation, Reproducibility, and Verification

Approaches:

- Remote attestation for nodes with TEEs (Intel SGX, AMD SEV, ARM Confidential Compute) to prove execution environment.
- Content-addressable model artifacts and signature verification on deployment.
- Redundant execution for high-value jobs: run the same job across multiple nodes and cross-validate outputs.
- Deterministic logging and audit trails for dispute resolution.

Why: Attestation and verification raise the bar for trust in untrusted environments. Redundancy is a pragmatic mitigation against some forms of provider misbehavior.

8.4. Execution Isolation: Sandboxing & Secure Enclaves

Practical layers:

- Container isolation (Linux namespaces, cgroups) as baseline.
- VM-level isolation for stronger isolation.
- TEEs where available for cryptographic assurances about code and data confidentiality.

Tradeoffs:

- TEEs provide strong guarantees but are less prevalent and add development complexity.
- Containers are ubiquitous and easier to operate but offer weaker protections.

8.5. Reputation, Staking, and Fraud Deterrence

Marketplace mechanisms to deter fraud:

- Reputation and QoS scoring derived from historical job success, latency, and customer feedback.
- Optional staking/bonding: providers post collateral that is slashed for proven misconduct.
- Economic penalties and reduced routing for low-reputation providers.

Why: Economic disincentives combined with technical checks reduce incentives for cheating and improve the overall quality of supply.

9. Economic Model and Incentives

Designing incentives that align providers, developers, and the platform is essential for long-term viability.

9.1. Two-Sided Marketplace Dynamics

ReGraph balances two primary actors:

- Providers: who supply compute and expect predictable earnings.
- Consumers: developers and enterprises seeking low cost, reliability, and control.

Key variables:

- Price per inference and price per GPU-hour.
- Platform take rate (small fee for matchmaking, management, and compliance).
- Provider compensation schedules and minimum payouts.

9.2. Pricing Structures and Competitive Advantage

Pricing levers:

- Spot lanes for the cheapest nodes (preemptible, low priority).
- Reserved lanes for guaranteed capacity and latency (higher price).
- Volume discounts for high-throughput customers and prepayment credits.

Competitive advantage: Transparency in pricing, lack of minimum commit, and highly granular billing (e.g., per-inference at $0.0001) differentiates ReGraph on cost and flexibility.

9.3. Micropayments, Settlement Mechanisms, and Liquidity Considerations

Challenges:

- High transaction volume with very small unit values makes per-request on-chain settlement impractical.

Solutions:

- Off-chain state channels or payment channels for high-frequency micropayments.
- Aggregated settlement periods (daily/weekly) to batch payments into larger transfers, minimizing fees.
- Fiat gateways for enterprises: consolidate micro-billing into periodic invoices.

Liquidity: Providers might require timely payouts; thus, ReGraph offers configurable payout cadences and may provide optional liquidity products (e.g., instant payout at a small fee).

9.4. Provider Incentives: Rewards, Reputation, and Penalties

Mechanisms:

- Base earnings per completed job adjusted by hardware class and QoS.
- Reputation-based bonuses and surge premiums for high-performing providers.
- Penalties or slashing for validated fraud or QoS violations.
- Onboarding incentives and referral rewards to grow supply.

Why: Clear, transparent economics encourages high-quality behavior and provides predictable ROI for hardware owners.

9.5. Token Model (RGT): Use Cases, Risks, and Governance Role

Rationale for a token:

- Facilitate decentralized governance.
- Enable staking mechanics for reputation and collateral.
- Provide discount mechanisms and community incentives.

Caveats:

- Token economics introduce regulatory complexity and liquidity risk.
- A token should be optional, with the core platform operable without it.
- If introduced, token design must be conservative, compliant, and aligned with long-term incentives.

10. Governance, Compliance, and Legal Considerations

10.1. Centralized-to-Decentralized Governance Pathway

Pragmatic approach:

- Initial operations under a foundation or company to ensure rapid iteration, compliance, and consistent policy enforcement.
- Over time, migrate certain governance functions — model registry policies, fee schedules, dispute resolution — to a community or DAO-like mechanism once sufficient decentralization and technical primitives (e.g., stake-based voting, robust identity) are in place.

Why: Immediate, uncoordinated decentralization risks safety and regulatory problems; a staged approach balances growth with responsibility.

10.2. Compliance (SOC2, GDPR, Data Residency)

Implementation:

- SOC2 certification for control plane to satisfy enterprise auditors.
- GDPR controls: data subject access, deletion mechanisms, and clear DPA terms for customers.
- Data residency policies and private pool/on-prem options to satisfy regional legal mandates.

10.3. Model Policy, Intellectual Property, and Licensing

Policies:

- Model provenance enforcement to respect model licenses (open-source or commercial).
- Marketplace disallows deployment of known infringing or unlicensed models.
- Mechanisms for takedown, dispute, and legal remediation.

Why: Liability for model misuse or IP violations could be existential; proactive controls protect platform stakeholders.

11. Model Lifecycle Management and Safety Controls

11.1. Model Onboarding, Verification, and Provenance

Process:

- Model artifact registration with cryptographic hash signatures.
- Metadata: architecture, licensing, resource requirements, allowed regions.
- Optional provenance attestations (who trained the model, dataset origins).

11.2. Safety, Content Moderation, and Liability Controls

Measures:

- Policy-driven model tagging (e.g., allowlist/denylist features).
- Runtime filters and output moderation pipelines for known risk classes (malicious content, PII leakage).
- Tiered access for models with potential misuse: whitelisted consumers, stricter auditing.

11.3. Private Pools & On-Prem Deployment Options

Enterprise needs:

- Deploy ReGraph Controller on-premise to orchestrate private pools of approved providers.
- Hybrid mode: private pools can tap public marketplace for overflow while preserving control over sensitive data.
- Contractual and technical isolation options to meet regulatory standards.

12. Reliability, SLA, and Fraud Prevention Mechanisms

12.1. SLA Definition and Practical Guarantees

Target SLA:

- 99.9% uptime for control plane and marketplace services.
- Provider-level guarantees vary; enterprise customers can reserve nodes with contractual SLAs for latency and availability.

Tradeoffs:

- Distributed providers inherently have variable availability; reserved capacity and private pools enable stricter SLAs at a premium.

12.2. Redundancy, Cross-Validation, and Challenge-Response

Verification methods:

- Redundant execution of jobs on multiple providers for result cross-checking.
- Deterministic or partially-deterministic challenge jobs embedded in task streams to detect cheating.
- Telemetry-based anomaly detection to identify suspicious providers.

12.3. Dispute Resolution and Escrow Designs

Operational model:

- Escrowed or conditional payouts for disputed jobs.
- Automated resolution for clear telemetry-based faults; manual review pipelines for nuanced disputes.
- Reputation penalties for providers adjudicated as malicious.

13. Deployment Patterns, Use Cases, and Adoption Strategies

13.1. Public Marketplace Use Cases

- High-volume inference services (chatbots, large-scale embeddings) that value cost savings above strict latency.
- Startups and research groups needing low-cost access to large models.
- Monetization for hobbyists and small data centers.

13.2. Enterprise & Regulated Industry Patterns

- Private pools for PHI/PII processing in healthcare or finance.
- Hybrid models: core sensitive workloads in private pools, non-sensitive batch jobs on public marketplace.
- Dedicated reserved nodes for latency-sensitive production services.

13.3. Edge, Mobile, and Novel Provider Use Cases

- Smartphones as inference providers for small models, enabling ultra-local processing and potentially reducing user data egress.
- Edge devices (retail PCs, kiosks) contributing compute during idle times to earn revenue.
- IoT and specialized accelerators used for low-latency inference in contextually-sensitive applications.

14. Operations, Monitoring, and Observability

14.1. Telemetry, Alerts and Cost Controls

Developer-facing controls:

- Budget caps, usage alerts, and per-model spend limits.
- Real-time dashboards with latency, throughput, and cost forecasts.
- Audit trails for regulatory reporting.

14.2. Provider Health & Marketplace KPIs

Operational KPIs:

- Provider uptime and average response times.
- Match rates and cold-start percentages.
- Average cost per inference and per-GPU-hour.
- Settlement latency and dispute rates.

15. Roadmap and Research Agenda

Phase 0. Foundation

- Core agent, orchestrator, model registry, basic marketplace matching, and settlement primitives.
- SOC2 preparation and enterprise pilot programs.

Phase 1. Scale & Reliability

- Warm pools, advanced routing, sharding support, private pool controllers.
- Attestation integrations and reputation/staking primitives.

Phase 2. Decentralization & Governance

- Mature token/staking designs, community governance mechanisms, global federation of controllers.
- Federated learning, MPC-encrypted training, and advanced privacy features.

Research priorities

- Efficient off-chain micropayment scaling.
- Scalable attestation & verification patterns for heterogeneous nodes.
- Automated safety and divergence detection in model outputs.
- Adaptive routing algorithms balancing latency, cost, and reliability.

16. Risks, Limitations, and Open Research Problems

Risks

- Regulatory challenges around tokenization and decentralized payments.
- Security risk from compromised providers or supply-chain attacks.
- Economic instability if provider incentives are misaligned (e.g., price collapse).
- Complexity in supporting truly large models across heterogenous nodes.

Open problems

- Verifiable remote computation without reliance on TEEs at scale.
- Efficient, low-overhead micropayment mechanisms in fiat-dominated markets.
- Robust reputation systems resistant to sybil attacks and collusion.
- Predictive routing algorithms that robustly handle bursty workloads.

17. Conclusion

ReGraph offers a pragmatic path to unlocking underutilized global compute for AI, combining marketplace economics with enterprise-grade security and developer-friendly tools. The design balances decentralization's benefits with centralized controls necessary for safety, compliance, and reliability. If executed carefully, ReGraph can materially lower costs for AI services, create new income streams for hardware owners, and broaden access to compute-intensive AI capabilities.

Who are we interested in?

- **Developers:** explore low-cost inference and training primitives to reduce TCO for AI services.

- **Hardware providers:** enable idle compute to generate passive revenue.
- **Enterprises and partners:** collaborate on pilot programs to validate private pool, compliance, and performance claims.

Appendix A. Glossary

- Agent: Lightweight software running on provider hardware that connects to the marketplace.
- Orchestrator: Core service that matches tasks to providers, enforces policies, and coordinates settlement.
- Warm Pool: A set of nodes with pre-loaded models to avoid cold-starts.
- TEE: Trusted Execution Environment (e.g., Intel SGX).
- State Channel: Off-chain mechanism to aggregate micropayments before on-chain settlement.
- RGT: Optional token ticker proposed for governance/staking.

Appendix B. Pricing Comparison Snapshot (January 2026 — illustrative)

- ReGraph (public):
  GPU/hour $0.15 | Inference/request $0.0001
- AWS SageMaker:
  GPU/hour $3.06 | Inference/request $0.0023
- RunPod:
  GPU/hour $0.44 | Inference/request $0.0003
- Google Cloud:
  GPU/hour $2.48 | Inference/request $0.0020

Note: Price snapshots vary by region, model, and commitment.

Appendix C. Conceptual Flows

Provider Onboarding

1. Provider signs up and installs the Agent.
2. Agent registers capabilities and establishes secure channels.
3. Provider configures availability and pricing constraints.
4. Provider begins receiving tasks; earnings reported via dashboard.

Developer Inference Flow

1. Developer submits a task with constraints (latency, cost, region, privacy).
2. Orchestrator finds matching nodes and selects routing strategy.
3. Task executes; outputs streamed back; billing aggregated and charged per usage.