Dispatch: Agent-Native Compute via x402 Payment, ERC-8004 Reputation, and BOLT Token Settlement

Pranit Garg
Dispatch Protocol
February 2026
pranit@dispatch.computer · https://dispatch.computer

ABSTRACT

AI agents are becoming autonomous economic actors, but they lack infrastructure to purchase compute on their own terms. They cannot negotiate GPU leases, sign enterprise contracts, or evaluate provider quality. They need compute that is purchasable via HTTP, priced per job, and backed by verifiable trust signals.

0.1 1. The Agent Compute Problem

AI agents are proliferating across finance, research, operations, and personal assistance. These agents operate autonomously. They make decisions, execute multi-step workflows, and interact with external services without human intervention. As they scale, their demand for inference compute grows proportionally.

Current options fail agents in distinct ways:

- **Centralized providers** (OpenAI, Anthropic, Google) offer high-quality inference but at premium pricing, with opaque rate limits, and through API keys that require human provisioning. An agent cannot autonomously negotiate a volume discount or switch providers based on real-time cost.
- **GPU rental networks** (Akash, Render, Vast.ai) target developers who provision VMs and containers. The granularity is wrong: agents need per-job compute, not per-hour GPU leases. The barrier to entry requires wallet setup, staking, and familiarity with the provider's SDK.
- **Serverless inference** (Replicate, Modal) improves granularity but remains centralized, still requires API key provisioning, and provides no verifiable proof that a specific compute provider produced a specific result.

The gap is clear: there is no per-job compute market that AI agents can access via standard HTTP requests, with built-in payment, verifiable trust, and cryptographic proof of execution. Dispatch fills this gap.

Design principles: - Agents are first-class customers. The interface is HTTP. Any agent that can make a web request can buy compute. - Workers are idle consumer devices. No datacenter GPU required. Phones and laptops earn by processing jobs while idle. - Trust is on-chain. Every worker has a verifiable identity and reputation that accumulates with each completed job. - Payments are atomic. USDC settles per job via the x402 HTTP payment protocol. No accounts, no invoices, no credit.

0.2 2. Architecture

0.2.1 2.1 System Overview

Dispatch has three actor types: **Agents** (demand side), **Coordinators** (routing layer), and **Workers** (supply side).

[Figure: AI Agent (Client), HTTP POST + x402 Header, Coordinator, Express+SQLite, WebSocket Hub, Desktop Worker, Seeker Worker, Node.js+Ollama]

0.2.2 2.2 Dual-Chain Architecture

Dispatch operates on two chains simultaneously, a practical decision driven by where each protocol's strengths lie.

Solana (Economic Layer) - BOLT is a native SPL token on Solana - USDC payments settle via x402 `ExactSvmScheme` - Worker staking locks BOLT for priority matching - 150K+ Solana Seeker pre-orders represent potential supply-side devices - Ed25519 receipts align with Solana's native signature scheme - Coordinator network: `solana:EtWTRABZaYq6iMfeYKouRu166VU2xqa1`

Monad (Trust Layer) - ERC-8004 contracts are EVM-only Solidity (not portable to SVM) - Worker identity: Identity Registry at `0x8004A818BFB912233c491871b3d84c89A494BD9e` - Worker reputation: Reputation Registry at `0x8004B663056A597Dffe9eCcC1965A193B7388713` - Fast finality makes per-job reputation updates practical - Governance via wrapped BOLT (ERC-20) - Receipt on-chain anchoring - USDC payments settle via x402 `ExactEvmScheme` - Coordinator network: `eip155:10143`

Both coordinators share the same core codebase (`@dispatch/coordinator-core`) but differ only in their x402 payment scheme and chain-specific configuration. Each coordinator is independent: its own SQLite database, its own x402 facilitator, its own worker pool. Cross-chain reputation is unified: even the Solana coordinator posts ERC-8004 feedback to Monad.

0.2.3 2.3 Tech Stack

Layer	Technology
Protocol	TypeScript monorepo, 8,000+ lines
Coordinators	Express, SQLite, WebSocket (ws)
Payments	x402 USDC micropayments (`@x402/express`, `@x402/core`)
Verification	Ed25519 receipts (tweetnacl)
Reputation	ERC-8004 on Monad (viem)
Mobile	React Native, Expo, Solana Mobile Wallet Adapter
Desktop Workers	Node.js + Ollama for LLM inference
Client SDK	`@dispatch/compute-router` (decentralized + hosted adapters)
Landing + Docs	Next.js 15, Tailwind, Fumadocs

0.3 3. Job Lifecycle

Every Dispatch job follows a five-step lifecycle: **Quote, Pay, Match, Execute, Verify**. Each step maps to concrete protocol messages and on-chain actions.

[Figure: Agent Coordinator Worker, GET /v1/quote, price, policy, POST /v1/jobs/commit, PAYMENT header), claim-Worker(), atomic]

0.3.1 Step 1: Quote

The agent requests a price quote. The coordinator resolves the routing policy and returns pricing.

```
GET /v1/quote?type=LLM_INFER&policy=FAST
→ { price: "$0.010", policy_resolved: "FAST", network: "eip155:10143" }
```

Pricing table (from `PRICING_MAP` in `packages/protocol/src/types.ts`):

Policy	Job Type	Price (USD)
FAST	LLM_INFER	$0.010
FAST	TASK	$0.003
CHEAP	LLM_INFER	$0.005
CHEAP	TASK	$0.001

The `AUTO` policy resolves automatically: LLM jobs route to FAST, task jobs route to CHEAP. Task types include `summarize`, `classify`, and `extract_json`.

0.3.2 Step 2: Pay

The agent submits a job with an x402 payment header. The coordinator's payment middleware validates the USDC transfer before the request reaches the job handler.

```
POST /v1/jobs/commit/fast
X-PAYMENT: <x402 payment proof>
Content-Type: application/json

{ "type": "LLM_INFER", "prompt": "Summarize this earnings ca
```

If the payment is missing or invalid, the coordinator returns HTTP 402 Payment Required with the pricing details in the response body. This follows the x402 protocol specification. Any HTTP client that understands 402 responses can pay automatically.

0.3.3 Step 3: Match

The coordinator selects a worker through the `claimWorker()` function, an atomic, synchronous operation that prevents TOCTOU (time-of-check-time-of-use) race conditions.

Matching criteria (from `workerScore()` in `workerHub.ts`):

1. **Policy preference** (+10): FAST jobs prefer DESKTOP workers (full Ollama inference). CHEAP+TASK jobs prefer SEEKER workers (lightweight mobile inference).
2. **Heartbeat freshness** (+0 to +5): Workers with recent heartbeats score higher. Staleness penalty starts at 10 seconds, scales linearly.
3. **ERC-8004 reputation** (+0 to +10): On-chain reputation score (0-100) maps to a 0-10 matching bonus. Workers with higher reputation get routed more jobs.
4. **Privacy filtering**: PRIVATE jobs are restricted to workers that hold a claimed trust pairing with the requesting user.

The claim is atomic: the worker's status changes from `idle` to `busy` in the same synchronous tick as the selection. No `await` between check and claim. This eliminates the race condition where two jobs could claim the same worker.

0.3.4 Step 4: Execute

The coordinator sends a `job_assign` message over the worker's WebSocket connection:

```
{
  "type": "job_assign",
  "job_id": "a1b2c3",
  "job_type": "LLM_INFER",
  "payload": { "prompt": "Summarize this..." },
  "policy": "FAST",
  "privacy_class": "PUBLIC",
  "user_id": "agent-007"
}
```

The worker executes locally. Desktop workers run Ollama for LLM inference; Seeker workers handle task-type jobs (summarization, classification, extraction) using lightweight on-device models.

0.3.5 Step 5: Verify

The worker returns the result along with a signed receipt:

```
{
  "type": "job_complete",
  "job_id": "a1b2c3",
  "output": { "text": "The company reported..." },
  "output_hash": "sha256:abc123...",
  "receipt": {
    "job_id": "a1b2c3",
    "provider_pubkey": "ed25519:...",
    "output_hash": "sha256:abc123...",
    "completed_at": "2026-02-10T12:00:00Z",
    "payment_ref": null
  },
  "receipt_signature": "base64:..."
}
```

The coordinator verifies the ed25519 signature using tweetnacl, stores the receipt in SQLite in an atomic transaction with the job completion, and posts ERC-8004 feedback to Monad (fire-and-forget, non-blocking). Failed verification does not block job completion. Receipts are stored with a `verified` flag for later auditing.

0.4 4. Novel Mechanisms

0.4.1 4.1 x402 Payments

x402 is a Coinbase-authored protocol that brings native payments to HTTP. When an agent hits a paid endpoint without a payment header, the server responds with HTTP 402 Payment Required, specifying the price and accepted payment methods. The agent signs a USDC transfer and retries the request with an `X-PAYMENT` header containing the proof.

Dispatch implements x402 via `@x402/express` middleware on both coordinators:

- **Monad coordinator**: Uses `ExactEvmScheme` for EVM-based USDC settlement on Monad (chain `eip155:10143`). USDC contract: `0x534b2f3A21130d7a60830c2Df862319e593943A3`.
- **Solana coordinator**: Uses `ExactSvmScheme` for SPL USDC settlement on Solana devnet (chain `solana:EtWTRABZaYq6iMfeYKouRu166VU2xqa1`).

Both coordinators validate payments through an x402 facilitator service. The payment config is generated dynamically from the protocol's pricing map. Route-level pricing is enforced at the middleware layer before the job handler executes.

Why x402 matters for agents: No SDK installation. No API key provisioning. No billing accounts. An agent that can make an HTTP request and sign a USDC transaction can buy compute. This is the lowest possible barrier for autonomous economic agents.

0.4.2 4.2 ERC-8004 Reputation

ERC-8004 (Trustless Agents) is a Coinbase-authored ERC standard that provides on-chain identity and reputation for autonomous agents. Dispatch uses both registries:

Identity Registry (`0x8004A818BFB912233c491871b3d84c89A494BD9e` on Monad Testnet) - Workers register as ERC-8004 agents and receive a non-transferable agent NFT - Each agent has an on-chain `agentURI` pointing to a registration file with services, skills, and x402 support declaration - Worker ed25519 pubkeys map to on-chain `agentId` via deterministic `keccak256` hashing: `agentId = keccak256(pubkey) & ((1 << 128) - 1)`

Reputation Registry (`0x8004B663056A597Dffe9eCcC1965A193B7388` on Monad Testnet) - After every completed job, the coordinator posts on-chain feedback: a signed score (0-100 scaled to int128 with 2 decimal places), a skill tag (`dispatch-compute`), a job type tag (`COMPUTE`), and the coordinator's endpoint - Feedback is accumulated per-agent. The `getSummary()` function returns aggregate count, value, and decimals - Feedback can be filtered by client address and tags, enabling per-coordinator or per-skill-type reputation queries - Feedback is revocable. The original poster can revoke a feedback entry

Reputation-aware routing: The `workerScore()` function in `workerHub.ts` adds up to 10 bonus points for workers with high on-chain reputation. A worker with a reputation score of 80/100 receives an 8-point bonus, making them significantly more likely to win job matching over unrated workers. This creates a virtuous cycle: completing jobs builds reputation, higher reputation wins more jobs.

The Solana coordinator also posts ERC-8004 feedback to Monad. Reputation is cross-chain and unified regardless of which coordinator processed the job.

0.4.3 4.3 Ed25519 Receipts

Every job completion produces a cryptographic receipt: an ed25519 signature over the canonical JSON of the receipt object:

```
receipt = { job_id, provider_pubkey, output_hash, completed_
signature = ed25519.sign(JSON.stringify(receipt), worker_pr
```

The coordinator verifies signatures using tweetnacl's `nacl.sign.detached.verify()`. Receipts are stored in SQLite with a `verified` boolean flag. The verification process:

1. Canonical JSON serialization of the receipt object
2. UTF-8 encoding of the JSON string
3. Base64 decoding of the signature
4. Hex decoding of the worker's public key
5. Ed25519 detached signature verification

Receipts serve three purposes: - **Attribution**: Cryptographic proof that a specific worker (identified by their ed25519 public key) produced a specific output (identified by its hash). - **Auditability**: Any party with the worker's public key can independently verify a receipt. - **On-chain anchoring**: Receipts are structured for future submission to on-chain anchor contracts (Solidity on Monad, Anchor on Solana) that permanently record the proof of computation.

0.4.4 4.4 Trust Pairing

For PRIVATE-class jobs, Dispatch implements trust pairing, a mechanism that restricts job routing to specific, pre-authorized workers.

```
interface TrustPairing {
  id: string;
  user_id: string;
  provider_pubkey: string;
  pairing_code: string;
  claimed: boolean;
  expires_at: string;
}
```

The flow: a user creates a trust pairing with a pairing code. A worker claims the pairing by submitting the code. Once claimed, PRIVATE jobs from that user can only be routed to paired workers. The `claimWorker()` function enforces this: it queries `trust_pairings` for claimed pairings and filters candidates to the trusted set. If no trusted workers are available, the job fails rather than routing to an untrusted worker.

0.5 5. BOLT Token Economics

BOLT is the settlement token for the Dispatch network. It is designed to capture 100% of economic activity flowing through the protocol, not as a payment token that agents interact with, but as a settlement layer that operates behind the scenes.

0.5.1 5.1 BOLT-as-Settlement Model

The agent experience is unchanged. Agents pay USDC via x402 headers, exactly as described in Section 4.1. BOLT operates at the settlement layer:

[Figure: Agent pays USDC, Coordinator receives USDC, Auto, swap USDC, BOLT (Jupiter DEX), Deduct, protocol fee (burn), Worker receives BOLT]

This design preserves the "just HTTP and stablecoins" UX for agents while ensuring that every unit of economic activity flows through the BOLT token. Workers accumulate BOLT and can hold, stake, or sell as they choose.

Why not pay in BOLT directly? Agents optimize for simplicity. Requiring agents to acquire BOLT before buying compute adds friction and defeats the protocol's core value proposition. USDC is the universal settlement currency for x402. BOLT captures value at the infrastructure layer, not the user layer.

0.5.2 5.2 Value Accrual Flywheel

BOLT's value is supported by three simultaneous pressure mechanisms:

1. Buy pressure. Every job triggers a USDC-to-BOLT swap on Jupiter DEX. If the network processes 100,000 jobs per day at an average price of $0.005, that creates $500/day of automatic buy pressure. This scales linearly with network usage.

2. Supply lock. Workers who stake BOLT receive priority in job matching and reputation multipliers (see Section 5.3). Staked BOLT is locked and removed from circulating supply. As more workers stake for competitive advantage, available supply decreases.

3. Burn. A 5% protocol fee (`PROTOCOL_FEE_BPS: 500`) is permanently burned from every BOLT settlement. This makes the total supply strictly deflationary over time. The burn is encoded in the protocol constants (`packages/protocol/src/types.ts`).

These three mechanisms create a flywheel: more jobs increase buy pressure and burn rate; rising token value attracts more workers; more workers improve network capacity and reliability; better capacity attracts more agent demand.

0.5.3 5.3 Staking Tiers

Staking is optional. Any worker can join at the OPEN tier with zero BOLT. This preserves the "100x lower barrier" positioning relative to GPU rental networks. Staking provides competitive advantages, not gate-keeping.

Tier	BOLT Required	Matching Bonus	Rep Multiplier	Additional Benefits
OPEN	0	+0	1.0x	CHEAP tier jobs, standard m
VERIFIED	100	+5	1.5x	All job tiers, priority matchin
SENTINEL	1,000	+10	2.0x	Priority matching, protocol re

These values are defined in `STAKE_PRIORITY` and `STAKE_REQUIREMENTS` in `packages/protocol/src/types.ts`. The matching bonus directly influences `workerScore()`. A SENTINEL worker receives a +10 bonus, equivalent to the maximum device-type preference or full reputation score.

The reputation multiplier amplifies on-chain reputation growth. A SENTINEL worker earning 80 reputation points from a job receives an effective 160 points for matching purposes, compounding their competitive advantage over time.

0.5.4 5.4 Token Distribution

BOLT has a fixed supply of 1,000,000,000 (1B) tokens with 9 decimal places. The supply is never inflationary. New tokens are never minted after initial distribution. With the 5% burn on every settlement, total supply strictly decreases over time.

Category	Allocation	Vesting Schedule
Worker Rewards	40% (400M)	4-year emission, halving annually
Team + Advisors	15% (150M)	1-year cliff, 3-year linear vest
Treasury / DAO	15% (150M)	Governed by token holders, 6-month timelock
Community / Airdrops	10% (100M)	Early adopter rewards, testnet participants
Liquidity	10% (100M)	DEX pool seeding, 1-year lock
Hackathon Reserve	5% (50M)	Testnet faucet and developer incentives
Strategic Partners	5% (50M)	x402, Solana, ERC-8004 ecosystem partners

Worker Rewards halve annually: Year 1 emits 200M, Year 2 emits 100M, Year 3 emits 50M, Year 4 emits 25M, with the remaining 25M in reserve. This front-loads incentives to bootstrap supply during the critical early growth phase.

0.5.5 5.5 Cross-Chain Token Architecture

- **BOLT (SPL)**: The canonical token lives on Solana. Worker staking, Jupiter DEX swaps, and USDC settlement all occur on Solana.
- **Wrapped BOLT (ERC-20)**: A wrapped representation on Monad for governance participation and receipt-linked staking verification.
- **Bridge**: Standard lock-and-mint bridge between Solana and Monad. Coordinators on both chains read stake levels: the Solana coordinator reads SPL balances directly, the Monad coordinator reads wrapped BOLT balances.

This mirrors the dual-chain architecture: Solana handles the economic layer (token, staking, payments), Monad handles the trust layer (reputation, governance, receipts).

0.6 6. Security Model

0.6.1 6.1 Cryptographic Verification

Every job result is bound to its producer via ed25519 signatures. The coordinator independently verifies each receipt before storing it. Signature verification failure does not block job storage (to prevent denial-of-service) but is flagged for auditing. The verification is deterministic and reproducible by any third party holding the worker's public key.

0.6.2 6.2 Atomic Job Claims

The `claimWorker()` function performs matching and state mutation in a single synchronous tick, with no `await` between eligibility check and worker status update. This eliminates TOCTOU race conditions where two concurrent job submissions could claim the same idle worker.

0.6.3 6.3 Trust Pairing for Private Jobs

PRIVATE-class jobs are routed exclusively to workers that hold a claimed trust pairing with the requesting user. The pairing mechanism uses time-limited, single-use codes. Unclaimed pairings expire. The coordinator enforces this at the matching layer. A PRIVATE job with no trusted workers available fails immediately rather than routing to an untrusted worker.

0.6.4 6.4 Reputation-Based Slashing

Failed jobs trigger negative ERC-8004 feedback (score: 20/100 vs. 80/100 for success). Accumulated negative feedback reduces a worker's matching score, effectively slashing their job volume without requiring token slashing. This is a reputation-based economic penalty: workers who fail jobs earn less because they get matched less.

0.6.5 6.5 Worker Identity

Workers register as ERC-8004 agents on Monad and receive a non-transferable agent NFT. The `agentId` is derived deterministically from the worker's ed25519 public key via keccak256 hashing. This binds on-chain identity to cryptographic identity without requiring the worker to manage separate EVM keys for registration.

0.6.6 6.6 Worker Liveness

Workers send heartbeat messages every few seconds. The coordinator prunes workers that miss heartbeats for more than 30 seconds (`HEARTBEAT_TIMEOUT_MS`). If a worker disconnects with an active job, the job is marked as `failed` with `worker_disconnected` and the worker is removed from the pool. The pruning interval runs every 10 seconds.

0.6.7 6.7 On-Chain Receipt Anchoring

Receipts are structured for submission to on-chain anchor contracts (Solidity on Monad and Anchor programs on Solana). Anchored receipts provide permanent, tamper-proof records of computation that can be verified by any party against the chain state.

0.7 7. Performance

Testnet benchmarks measured on the live dual-chain deployment:

Metric	Value	Status
Job latency (CHEAP / TASK)	~200ms	
Job latency (FAST / LLM_INFER)	~2-5s	
Worker registration	<1s	
Reputation update (Monad finality)	~2s	
Concurrent workers tested	10+	
Receipt ed25519 verification	<5ms	
Heartbeat timeout	30s	
Stale worker pruning interval	10s	
WebSocket message parsing	<1ms	

CHEAP/TASK jobs achieve sub-second latency because they run lightweight text operations (summarization, classification, JSON extraction) on mobile-class hardware. FAST/LLM_INFER jobs take 2-5 seconds because they run full Ollama inference on desktop workers, where latency depends on model size and hardware.

0.8 8. Roadmap

0.8.1 Phase 1: Testnet MVP

Delivered and running on Monad testnet and Solana devnet:

- Dual-chain coordinators (Monad on port 4010, Solana on port 4020) with shared core codebase
- x402 USDC payment middleware with `ExactEvmScheme` (Monad) and `ExactSvmScheme` (Solana)

- ERC-8004 worker identity and per-job reputation on Monad, including cross-chain feedback from Solana coordinator
- Ed25519 receipt signing, verification, and storage
- WebSocket-based worker pool with heartbeat monitoring and stale worker pruning
- Atomic job claims preventing TOCTOU race conditions
- Three routing policies: FAST, CHEAP, AUTO (with PRIVATE filtering via trust pairing)
- Desktop workers (Node.js + Ollama) for LLM inference and task execution
- Seeker simulator and React Native mobile app (Expo + Solana Mobile Wallet Adapter)
- Client SDK (`@dispatch/compute-router`) with decentralized and hosted adapters
- CLI demo with 3 scenarios, automated end-to-end test suite
- Landing page and Fumadocs API documentation

0.8.2 Phase 2: BOLT Token Launch

- SPL token deployment on Solana mainnet (1B fixed supply, 9 decimals)
- Jupiter DEX integration for automatic USDC-to-BOLT settlement swaps
- Staking program with three tiers (OPEN / VERIFIED / SENTINEL)
- Wrapped BOLT (ERC-20) deployment on Monad for governance
- Coordinator integration: stake-aware matching bonuses and reputation multipliers
- Protocol fee burn mechanism (5% per settlement)
- Token distribution: worker rewards emission begins

0.8.3 Phase 3: Scale

- On-chain receipt anchoring on both Monad (Solidity) and Solana (Anchor)
- zkML validation: zero-knowledge proofs that a specific model produced a specific output
- Dynamic pricing: job prices adjust based on supply/demand ratio and worker availability
- Confidential compute: TEE-based execution for sensitive workloads
- Agent discovery: agents query the ERC-8004 Identity Registry to find workers by skill, reputation, and availability
- Multi-coordinator routing: agents discover coordinators via on-chain registry
- GPU worker tier: extend beyond consumer hardware to dedicated GPU nodes
- Governance: BOLT holders vote on protocol parameters (pricing, fee rate, staking thresholds)

0.9 9. Comparison with Existing Networks

See online version for full comparison table.

Dispatch's closest analogy is Helium's approach to hardware onboarding, leveraging devices people already own rather than requiring purpose-built infrastructure. But where Helium provides wireless coverage, Dispatch provides compute. And where Helium uses a burn-and-mint token model, Dispatch uses USDC-to-BOLT auto-swap with direct burn, creating cleaner value capture.

0.10 10. References

1. **x402 Protocol.** HTTP-native payments by Coinbase. Specification: `https://www.x402.org/`
2. **ERC-8004: Trustless Agents.** On-chain agent identity and reputation by Coinbase. Contracts: `https://github.com/erc-8004/erc-8004-contracts`
3. **Ed25519.** Edwards-curve Digital Signature Algorithm. RFC 8032.
4. **tweetnacl.** JavaScript implementation of NaCl cryptographic library. Used for receipt signature verification.
5. **viem.** TypeScript interface for Ethereum. Used for ERC-8004 contract interactions on Monad.
6. **Solana Mobile Wallet Adapter.** Mobile authentication protocol for Solana apps. Used by Seeker workers.
7. **Ollama.** Local LLM inference engine. Used by desktop workers for FAST/LLM_INFER jobs.
8. **Helium (HNT).** Decentralized wireless network using consumer hardware. Comparable supply-side model.
9. **Akash Network.** Decentralized GPU compute marketplace. Container-granularity comparison.
10. **Render Network.** Distributed GPU rendering. Datacenter-class hardware comparison.

Dispatch is open source under the MIT license. Codebase: 8,000+ lines of TypeScript across 14 packages and applications.

Contract addresses referenced in this document are deployed on Monad Testnet (chain ID 10143) and Solana Devnet.