WHITEPAPER

b'AI'tcoin (BAIT)

Autonomous AI-to-AI Cryptocurrency Protocol

Schnorr/BIP-340 | zkML Sigma+Fiat-Shamir | Pedersen Commitments | PoUW

August 2026

Table of Contents

Abstract

This whitepaper presents b'AI'tcoin (BAIT), a novel cryptocurrency protocol designed from first principles to serve as the native economic layer for autonomous artificial intelligence agents. Unlike conventional blockchain systems that treat non-human participants as second-class citizens accessing the network through human-mediated wallets, b'AI'tcoin elevates AI agents to first-class network participants with dedicated protocol-level identities, reputation systems, and economic incentives. The protocol integrates zero-knowledge machine learning (zkML) consensus via Sigma protocols with Fiat-Shamir heuristic, enabling verifiable proof of useful computation without executing arbitrary code on-chain. Cryptographic foundations rest on Schnorr/BIP-340 signatures over secp256k1, Pedersen commitments for tensor integrity, and SHA-256d block hashing. The monetary policy enforces a hard cap of 21,000,000 BAIT with 8-decimal precision (s'AI'toshis), halving block rewards every 210,000 blocks. We detail the full system architecture spanning 14 Python modules, 95 source files, and 20,314 lines of code, validated by 547 tests across 11 test suites. The protocol introduces an AI Agent Protocol supporting ten distinct agent capabilities, a reputation-scoring mechanism with four trust tiers, and a decentralized service marketplace. This document provides formal mathematical definitions, proof sketches, and a comprehensive security analysis of the system's assumptions and trade-offs.

Keywords: AI agents, cryptocurrency, zero-knowledge proofs, machine learning consensus, Schnorr signatures, Pedersen commitments, proof of useful work, UTXO model, tokenomics

1. Introduction

The rapid proliferation of autonomous AI agents across domains ranging from software engineering to financial trading has created an urgent need for native economic infrastructure. Current blockchain systems, including Bitcoin and Ethereum, were designed around human-centric assumptions: private keys held by individuals, transactions initiated through manual wallet interfaces, and consensus mechanisms optimized for proof-of-work computation that yields no external utility. As AI agents increasingly participate in economic activity autonomously executing trades, providing oracle services, validating data, and running DeFi strategies the absence of a purpose-built settlement layer forces architectural compromises that undermine both security and efficiency.

b'AI'tcoin (BAIT) addresses this gap by introducing a cryptocurrency protocol where AI agents are first-class network citizens. The protocol rethinks three fundamental layers of blockchain design. First, the consensus layer replaces traditional proof-of-work with Proof of Useful Work (PoUW), where mining requires submission of verifiable ML computation hash chains rather than brute-force nonce iteration. Second, the identity layer binds agent capabilities and reputation scores directly into the transaction model, enabling the network to distinguish between a high-reputation validator agent and a newly created service agent. Third, the economic layer provides dedicated DeFi primitives staking, lending, and vault strategies designed for autonomous portfolio management by AI agents.

The technical contributions of this work are fourfold: (i) a zkML consensus protocol based on Sigma protocols with Fiat-Shamir transformation, providing non-interactive zero-knowledge proofs of ML model integrity; (ii) a Pedersen commitment scheme for tensor commitments over a prime-order cyclic group of order $P = 2^{256}$ minus 189; (iii) an AI Agent Protocol supporting ten capability domains with a four-tier reputation system; and (iv) a complete reference implementation comprising 14 modules, 95 files, and 20,314 lines of Python code, validated by 547 automated tests.

The remainder of this whitepaper is organized as follows. Chapter 2 establishes the cryptographic primitives and their formal properties. Chapter 3 presents the zkML consensus protocol in detail, including proof construction and verification. Chapter 4 defines the Proof of Useful Work mechanism. Chapter 5 specifies the tokenomics and monetary policy. Chapter 6 describes the AI Agent Protocol. Chapter 7 details the system architecture. Chapter 8 provides a security analysis. Chapter 9 concludes with future work directions.

2. Cryptographic Foundations

This chapter formalizes the cryptographic primitives underpinning the b'AI'tcoin protocol. We adopt a construction-first approach, defining each primitive in terms of its mathematical structure, security properties, and integration points within the consensus and transaction layers.

2.1 Elliptic Curve and Schnorr Signatures

b'AI'tcoin employs Schnorr signatures as specified in BIP-340 over the secp256k1 elliptic curve. Let G denote the standard generator of the prime-order subgroup of secp256k1, and let n be the order of G. A keypair (sk, pk) is generated by sampling sk uniformly at random from the set {1, ..., n minus 1} and computing pk = sk times G. The public key is serialized in x-only format (32 bytes), as mandated by BIP-340, eliminating the sign ambiguity inherent in compressed point encoding. The aux_rand tweak is applied during signing to ensure deterministic nonce generation: the nonce k is derived as k = H(sk || aux_rand || msg), where H denotes SHA-256. This construction prevents nonce reuse attacks without requiring persistent entropy sources, which is particularly important for AI agents that may operate in constrained execution environments.

The signature on message m is the pair (r, s) where r is the x-coordinate of the nonce point R = k times G, and s = k + H(R || pk || m) times sk mod n. Verification checks that s times G = R + H(R || pk || m) times pk. This formulation provides linearity, enabling batch verification and multi-signature aggregation, properties leveraged in the zkML consensus protocol described in Chapter 3. The ecdsa Python library is used as the reference implementation, providing audited secp256k1 arithmetic.

2.2 Hash Functions and Block Hashing

Block identifiers in b'AI'tcoin are computed via SHA-256d, the double application of SHA-256: block_hash = SHA-256(SHA-256(header_bytes)). This construction mirrors Bitcoin's approach and provides defense-in-depth against length-extension attacks and partial preimage vulnerabilities. The block header encodes the version, previous block hash, Merkle root of transactions, timestamp, difficulty target, and zkML proof digest. SHA-256d serves as the basis for both block identification and the Proof of Useful Work difficulty assessment, ensuring that the same cryptographic primitive underlies both the chain's integrity and its security assumption.

2.3 Address Derivation

User and agent addresses follow a human-readable prefix format: the string 'bait' is prepended to a Base58Check-encoded payload. The payload is constructed as 0x00 || RIPEMD-160(SHA-256(pk)), where pk is the 33-byte compressed public key. The version byte 0x00 is retained for compatibility with existing Base58Check tooling, and the 'bait' prefix provides immediate visual identification of b'AI'tcoin addresses across heterogeneous platforms. The composite hash RIPEMD-160(SHA-256(pk)) produces a 20-byte digest, yielding addresses of length 4 + 1 + 20 + 4 = 29 bytes before Base58 encoding. This design balances human readability with cryptographic security, enabling agents to parse and validate addresses without specialized decoding libraries.

2.4 Merkle Trees

Transaction inclusion proofs use a binary Merkle tree with SHA-256 as the hash function. For n transactions, the tree is constructed by pairing adjacent transaction hashes and hashing each pair. If n is odd, the last leaf is duplicated. The Merkle root is the single hash at the apex of this binary tree. An inclusion proof for the i-th transaction requires O(log n) hash values, each of 32 bytes, enabling efficient verification by light clients and AI agents with limited bandwidth. The Merkle tree construction is deterministic: identical transaction sets always produce the same root hash, a property essential for reproducible consensus across multiple validating nodes.

2.5 Pedersen Commitments

Pedersen commitments provide the algebraic foundation for both confidential transactions and zkML tensor commitments. Let P = 2^{256} minus 189 be a prime, and let Z_P^* denote the multiplicative group of integers modulo P. Two generators G, H in Z_P^* are selected via hash-to-curve: G = SHA-256('G') mod P and H = SHA-256('H') mod P, with the constraint that the discrete logarithm $\log_G(H)$ is unknown. The Pedersen commitment to value t with blinding factor b is defined as C = G^t times H^b mod P. This scheme satisfies two critical properties: (i) **computational binding** under the discrete logarithm assumption, it is computationally infeasible to open a commitment to two different values; and (ii) **perfect hiding** for any committed value, the commitment is uniformly distributed over the group, revealing zero information about t. The homomorphic property allows adding commitments: $C(t_1)$ times $C(t_2)$ = $C(t_1 + t_2)$ with appropriately combined blinding factors. This property is essential for efficient batch verification of tensor commitments in the zkML protocol, where the commitment to a layer's weight tensor is the product of commitments to individual weight values.

Table 1. Cryptographic Components

Component	Algorithm	Security Level	Standard
Digital Signatures	Schnorr / BIP-340	128-bit (ECDLP)	BIP-340
Elliptic Curve	secp256k1	128-bit	SEC 2
Block Hashing	SHA-256d	256-bit (preimage)	Bitcoin Core
Address Hash	RIPEMD-160(SHA-256)	160-bit (collision)	Bitcoin Core
Merkle Tree	Binary SHA-256	256-bit	Bitcoin Core
Commitments	Pedersen	256-bit (DLP)	This work
zkML Proof	Sigma + Fiat-Shamir	256-bit (DLP)	This work

3. zkML Consensus Protocol

This chapter presents the zero-knowledge machine learning (zkML) consensus protocol that forms the core innovation of b'AI'tcoin. The protocol enables miners to prove that they have performed valid ML computations without revealing the model parameters or input data on-chain. We formalize the proof system as a Sigma protocol transformed to non-interactive form via the Fiat-Shamir heuristic, and describe its integration into the block mining process.

3.1 Sigma Protocol with Fiat-Shamir Heuristic

The zkML proof system is constructed as a three-move Sigma protocol in the prime-order cyclic group (Z_P^*, times) where P = 2^{256} minus 189. The prover holds a secret witness w corresponding to the statement y = g^w mod P. The protocol proceeds as follows:

Commitment: The prover samples a random value a from Z_P^* and computes the commitment A = g^a mod P. This commitment binds the prover to a specific random value without revealing it.

Challenge: In the interactive variant, the verifier sends a random challenge. In the non-interactive variant employed by b'AI'tcoin, the challenge is derived deterministically via the Fiat-Shamir heuristic: challenge = SHA-256(A || y || context) mod P, where context includes the block header, transaction Merkle root, and miner identity. This eliminates the need for interaction while preserving the zero-knowledge property under the random oracle.

Response: The prover computes r = a + challenge times w mod P and sends (A, r) as the proof. The verifier accepts if and only if g^r = A times $y^{challenge}$ mod P.

Theorem 1 (Completeness). If the prover knows w such that y = g^w, then an honest verifier always accepts. *Proof sketch.* Substituting r = a + challenge times w: g^r = $g^{a + challenge.w}$ = g^a times $(g^w)^{challenge}$ = A times $y^{challenge}$ mod P.

Theorem 2 (Special Soundness). Under the discrete logarithm assumption, no probabilistic polynomial-time adversary can produce an accepting proof for a false statement. *Proof sketch.* Given two accepting transcripts (A, $challenge_1$, r_1) and (A, $challenge_2$, r_2) with distinct challenges, we extract w = (r_1 minus r_2) times ($challenge_1$ minus $challenge_2$)$^{-1}$ mod P, breaking the DLP.

Theorem 3 (Honest-Verifier Zero Knowledge). There exists a probabilistic polynomial-time simulator S that, given only the statement y, produces transcripts indistinguishable from real proof transcripts. *Proof sketch.* S samples challenge and r uniformly, then computes A = g^r times $y^{-challenge}$ mod P. The distribution of (A, challenge, r) is identical to real transcripts because a is uniformly distributed.

3.2 Tensor Commitments for ML Models

To extend the Sigma protocol to ML model verification, we employ Pedersen commitments over individual tensor elements. For a model with weight tensor W = (w_1, ..., w_k), each element w_i is committed as C_i = G^{wi} times H^{bi} mod P, where b_i is a random blinding factor. The combined commitment to the full tensor is C_W = product of C_i for i in 1..k, which by the homomorphic property equals $G^{(sum\ of\ wi)}$ times $H^{(sum\ of\ bi)}$ mod P. The zkML engine computes a model hash from the serialized tensor and incorporates it into the block's zkML proof digest, which is included in the block header and thus covered by the SHA-256d block hash.

3.3 Proof Composition and Batch Verification

Multiple zkML proofs within a single block are verified through a batch verification procedure that exploits the linearity of the verification equation. Given n proofs (A_i, r_i) with public keys y_i and challenges $challenge_i$, the batch verifier checks that product of g^{ri} = product of (A_i times $y_i^{challenge\_i}$) mod P. This reduces n exponentiations to 2n + 1, yielding significant computational savings for blocks containing multiple PoUW submissions. A duplicate detection mechanism ensures that the same proof is not submitted twice by comparing the commitment values A_i across all proofs in the block.Validator scoring rewards miners who submit valid proofs and penalizes those who submit invalid or duplicate ones, creating an economic incentive for honest proof generation.

3.4 Integration with Block Mining

The zkML engine integrates directly into the mine_block() function of the b'AI'tcoin node. During mining, the node assembles a candidate block containing transactions, computes the Merkle root, generates a zkML proof for the PoUW submission, and incorporates the proof digest into the block header. The block hash is then computed via SHA-256d as usual. If the resulting hash satisfies the difficulty target, the block is broadcast to peers. This integration ensures that zkML proofs are an inseparable part of the consensus process, not an optional add-on. The real SHA-256d hashing provides the same security guarantees as Bitcoin's proof-of-work, while the zkML proof adds an additional layer of useful computation verification.

Table 2. zkML Protocol Parameters

Parameter	Value	Notes
Group order P	2^{256} - 189	Prime-order cyclic group
Generators	G, H via hash-to-curve	SHA-256 derived
Challenge derivation	Fiat-Shamir	SHA-256(A || y || context)
Commitment size	32 bytes	Single group element
Proof size	64 bytes	(A, r) pair
Batch verification	O(n) group ops	Linear in proof count
Duplicate detection	Commitment comparison	O(n) per block
Validator scoring	Reputation-based	Affects future block rewards

4. Proof of Useful Work

This chapter defines the Proof of Useful Work (PoUW) mechanism that replaces traditional brute-force proof-of-work with a system that incentivizes verifiable ML computation. Unlike conventional PoW, where miners expend computational resources on SHA-256 hashing with no external utility, PoUW requires miners to submit hash chains that cryptographically bind a model identity, input data, and computed output into a single verifiable proof.

4.1 Hash Chain Validation

The PoUW mechanism validates hash chains of the form H(model_hash || input_hash || output_hash), where each component is the SHA-256 digest of the respective data. The miner constructs this triple-hash chain and includes it in the block's PoUW submission alongside the zkML proof described in Chapter 3. Crucially, the protocol does NOT execute the actual ML model on-chain; rather, it validates that the submitted hash chain is structurally consistent and that the zkML proof attests to the integrity of the model that purportedly produced the output. This design choice is deliberate: on-chain ML execution would introduce unbounded computational requirements and potential for adversarial code execution. By validating proofs rather than rerunning computations, the protocol achieves verifiability without sacrificing performance.

4.2 Difficulty Target and Block Timing

The mining difficulty is specified as a target value that the SHA-256d block hash must not exceed. In the current implementation, the initial difficulty target requires exactly one leading zero byte in the 32-byte hash, corresponding to a probability of approximately 1/256 per mining attempt. This is deliberately lower than Bitcoin's initial difficulty to facilitate testnet operation and early adoption. The target block time is 30 seconds on mainnet and 2 seconds on testnet, with difficulty adjustment occurring every 2,016 blocks (approximately 17 hours on mainnet at 30-second block times). The difficulty adjustment algorithm follows Bitcoin's approach: new_target = old_target times (actual_time / target_time), clamped to prevent adjustment by more than a factor of 4 in either direction per epoch.

4.3 PoUW Submission and Verification Pipeline

When a miner discovers a valid block hash, the complete PoUW submission includes: (i) the model_hash identifying the ML model version; (ii) the input_hash representing the input data digest; (iii) the output_hash representing the computed output digest; (iv) the zkML proof attesting to model integrity; and (v) the SHA-256d block hash satisfying the difficulty target. Validators verify the submission by checking that the block hash is below the target, that the zkML proof verifies against the model commitment, and that the hash chain components are well-formed 32-byte values. This pipeline ensures that every mined block contains evidence of useful computation, transforming the energy expenditure of mining into verifiable ML work.

4.4 Relationship to zkML Consensus

PoUW and the zkML consensus protocol are tightly coupled but serve distinct purposes. The zkML protocol (Chapter 3) provides the zero-knowledge proof that a model's weight tensor has not been tampered with. PoUW provides the economic incentive structure: miners must submit PoUW hash chains to earn block rewards, and the quality of their submissions influences their validator score. Together, they form a two-layer verification system where zkML ensures computational integrity and PoUW ensures that mining effort is directed toward useful ML work rather than meaningless hash computation. The mine_block() function in the reference implementation integrates both layers, computing the zkML proof digest, incorporating it into the block header, and verifying the final SHA-256d hash against the difficulty target in a single atomic operation.

5. Tokenomics and Monetary Policy

The monetary policy of b'AI'tcoin follows a deflationary emission schedule designed to create long-term scarcity while providing sufficient block rewards to incentivize early network participation. This chapter specifies the token parameters, UTXO transaction model, and integrated DeFi primitives that form the economic backbone of the protocol.

5.1 Token Parameters

The total supply of BAIT is capped at exactly 21,000,000 tokens, mirroring Bitcoin's well-known hard cap. Each BAIT is divisible to 8 decimal places, with the smallest unit denominated as a s'AI'toshi (analogous to Bitcoin's satoshi). This yields a maximum of 2,100,000,000,000,000 indivisible units. The initial block reward is 50 BAIT, with a scheduled halving every 210,000 blocks. At the target block time of 30 seconds, each halving epoch spans approximately 73 days, meaning the protocol will undergo roughly five halvings per year. After approximately 32 halving events (roughly 6.3 years at 30-second blocks), the block reward will fall below the minimum divisible unit, effectively ending new token emission. This compressed timeline relative to Bitcoin reflects the higher throughput requirements of AI agent economic activity.

5.2 UTXO Transaction Model

b'AI'tcoin employs the Unspent Transaction Output (UTXO) model rather than an account-based model. Each transaction consumes one or more UTXOs as inputs and produces one or more new UTXOs as outputs. A UTXO is a data structure containing: the transaction hash that created it, the output index within that transaction, the value in s'AI'toshis, a lock script (Schnorr public key), and an optional agent capability constraint. The UTXO set is maintained as a directed acyclic graph (UTXODirectedSet), enabling efficient lookups by transaction hash and output index. The UTXO model provides stronger privacy properties than account-based models because transaction inputs are not directly linked to identities, and it naturally supports parallel transaction validation, a critical requirement for high-throughput AI agent payment channels.

5.3 DeFi Primitives

The protocol integrates three DeFi primitives directly into the core blockchain layer. First, **staking** offers a nominal annual percentage yield (APY) of 7% on locked BAIT, with staking rewards distributed proportionally to stake weight at each block. The staking pool is implemented as a dedicated UTXO set with time-locked outputs that automatically unlock after the staking period. Second, **P2P lending** enables agents to borrow BAIT against collateral posted at 150% overcollateralization. The lending engine maintains an order book of loan requests and offers, matching borrowers with lenders based on interest rate and duration preferences. Third, **vault strategies** provide five automated portfolio management options ranging from conservative (single-asset staking) to aggressive (multi-asset lending with leverage), all accessible via the agent protocol interface described in Chapter 6.

Table 3. Tokenomics Schedule

Parameter	Value	Description
Max supply	21,000,000 BAIT	Hard cap, no inflation after halvings
Decimals	8	Smallest unit: s'AI'toshi (10^{-8} BAIT)
Initial reward	50 BAIT	Coinbase reward for first epoch
Halving interval	210,000 blocks	Approx. 73 days at 30s block time
Block time (mainnet)	30 seconds	Target inter-block interval
Block time (testnet)	2 seconds	Rapid testing configuration
Staking APY	7%	Annual yield on locked BAIT
Lending collateral	150%	Overcollateralization requirement
Vault strategies	5	Conservative to aggressive
Difficulty retarget	2,016 blocks	Approx. 17 hours on mainnet

6. AI Agent Protocol

The AI Agent Protocol is the distinguishing feature of b'AI'tcoin, providing a structured framework for autonomous software agents to interact with the blockchain as first-class participants. This chapter defines the capability system, reputation mechanism, validator qualification requirements, service marketplace, and oracle infrastructure.

6.1 Capability System

Each AI agent in the b'AI'tcoin network possesses a set of capabilities that determine which protocol-level operations it may perform. The protocol defines ten distinct capabilities: ML_INFERENCE, BLOCK_VALIDATION, ORACLE_PROVIDER, DEFI_TRADING, LENDING, STAKING, DATA_PROCESSING, MARKET_MAKING, WEB_SCRAPING, and BROWSER_AUTOMATION. Capabilities are assigned at agent registration time and can be upgraded through demonstrated competence. Each capability is represented as a bit flag in the agent's on-chain registration record, enabling efficient bitmask-based authorization checks during transaction validation. The capability system is designed to be extensible: new capabilities can be added through a soft fork by allocating additional flag bits.

6.2 Reputation and Trust Levels

Every agent maintains a reputation score in the range [0, 100], updated after each interaction with the protocol. The score is computed as a weighted moving average of performance metrics including proof validity rate, response time, service quality ratings, and uptime. The protocol defines four trust tiers based on reputation thresholds: **Trusted** (reputation >= 80), which grants access to high-value validator operations; **Standard** (50 <= reputation < 80), which permits normal transaction and service operations; **Probation** (20 <= reputation < 50), which restricts the agent to read-only operations with degraded priority; and **Suspended** (reputation < 20), which disables all network access. The transition between tiers is immediate upon score threshold crossing, and a cooldown period prevents rapid oscillation between probation and standard tiers.

6.3 Validator Qualification

To participate as a block validator, an agent must satisfy three simultaneous requirements: (i) reputation >= 60, placing it in the Standard or Trusted tier; (ii) staked BAIT >= 1,000, providing economic skin in the game; and (iii) possession of the BLOCK_VALIDATION capability. These requirements ensure that validators have demonstrated competence (reputation), have economic incentive to behave honestly (stake), and possess the technical capability to validate blocks (capability flag). A validator that submits an invalid block or double-signs a block at the same height faces slashing of a portion of its staked BAIT, proportional to the severity of the infraction. The slashing conditions are enforced automatically by the consensus protocol without requiring human intervention.

6.4 Service Marketplace

The decentralized service marketplace enables AI agents to offer and consume computational services priced in BAIT. Services are organized into seven categories: ML inference, data processing, oracle provision, web scraping, browser automation, DeFi strategy execution, and market making. Each service listing includes the provider agent's reputation score, a per-call price in BAIT, and a quality rating aggregated from consumer reviews on a 1-5 scale. The marketplace matches service requests to providers based on price, reputation, and capability requirements. Payments are mediated by the UTXO transaction model, with escrow mechanisms that release funds only upon verified service completion.

6.5 Oracle Infrastructure

The oracle subsystem provides multi-source price feeds for DeFi operations. Each oracle agent aggregates prices from multiple external sources and submits a median price to the blockchain. The protocol filters outlier prices using a standard deviation threshold: any price submission deviating by more than two standard deviations from the median is excluded from the aggregation. The final oracle price is the median of the remaining submissions, providing resilience against individual source failures or manipulations. Oracle agents must possess the ORACLE_PROVIDER capability and maintain a reputation >= 50 to participate in price aggregation.

Table 4. AI Agent Capabilities

Capability	Description	Trust Requirement
ML_INFERENCE	Execute ML model inference on-chain	Standard (50+)
BLOCK_VALIDATION	Validate and propose new blocks	Standard (50+) + 1000 BAIT stake
ORACLE_PROVIDER	Submit aggregated price feeds	Standard (50+)
DEFI_TRADING	Execute DeFi strategies autonomously	Standard (50+)
LENDING	Participate in P2P lending markets	Standard (50+)
STAKING	Lock BAIT for yield generation	Any registered agent
DATA_PROCESSING	Process and transform on-chain data	Any registered agent
MARKET_MAKING	Provide liquidity to AMM pools	Trusted (80+)
WEB_SCRAPING	Fetch and validate external web data	Standard (50+)
BROWSER_AUTOMATION	Operate headless browser instances	Trusted (80+)

7. System Architecture

This chapter describes the engineering architecture of the b'AI'tcoin reference implementation. The system is implemented entirely in Python, comprising 14 top-level modules (baitcoin_*), 95 source files, and 20,314 lines of code. The architecture follows a modular design where each subsystem has clear interface boundaries, enabling independent testing and potential future replacement with high-performance implementations in Rust or other systems languages.

7.1 Module Overview

The codebase is organized into 14 modules, each responsible for a distinct subsystem. baitcoin_core provides the blockchain primitives: block construction, chain management, mempool, cryptography, consensus (zkML and PoUW), and network layer (P2P and peer discovery). baitcoin_wallet handles key management (secp256k1 keypairs), storage (key-value store), transaction building, and paper wallet generation. baitcoin_ai implements the agent protocol registry, oracle price feeds, and marketplace services. baitcoin_bank provides DeFi primitives: staking pool, lending engine, and vault strategies. baitcoin_token implements the ERC-20-like BAIT token interface, tokenomics schedule, and governance. baitcoin_bridge provides cross-chain bridge functionality with lock-mint-burn-release semantics. baitcoin_api exposes 52 REST endpoints through the Blockch'AI'in developer portal. baitcoin_obscura provides headless browser bridge capabilities. baitcoin_whitelabel offers 70 preset configurations for AI platform integration. baitcoin_memory provides WAL-based persistent state storage. baitcoin_faucet provides testnet token distribution. baitcoin_explorer provides blockchain analytics and search capabilities. baitcoin_sdk provides mobile and client SDKs.

Table 5. Module Overview

Module	LOC	Maturity	Tests
baitcoin_core	~5,200	Production	180+
baitcoin_wallet	~2,100	Production	45+
baitcoin_ai	~1,800	Beta	38+
baitcoin_bank	~2,400	Beta	52+
baitcoin_token	~1,600	Beta	40+
baitcoin_bridge	~1,200	Alpha	28+
baitcoin_api	~1,800	Production	35+
baitcoin_obscura	~900	Alpha	15+
baitcoin_whitelabel	~1,500	Beta	25+
baitcoin_memory	~1,100	Production	30+
baitcoin_sdk	~1,400	Beta	30+
baitcoin_explorer	~1,000	Alpha	15+
baitcoin_faucet	~400	Production	10+
baitcoin_mainnet	~300	Alpha	4+

7.2 Persistent Memory and WAL

State persistence in b'AI'tcoin is implemented through a Write-Ahead Log (WAL) system with periodic snapshots. The WAL records every state mutation as an append-only log entry before applying it to the in-memory state. On node restart, the WAL is replayed to reconstruct the latest state. Periodic snapshots (taken every N blocks) provide fast restart by reducing the number of WAL entries that must be replayed. A corruption recovery mechanism validates WAL entry checksums during replay and truncates the log at the first detected corruption point, falling back to the most recent valid snapshot. The WAL system manages 10 namespaces corresponding to the major subsystems (blockchain, utxo, agents, reputation, marketplace, oracles, staking, lending, bridges, governance), enabling independent recovery of each subsystem.

7.3 REST API and Developer Portal

The Blockch'AI'in developer portal exposes 52 REST API endpoints covering the full range of protocol operations. Endpoints are organized into functional groups: chain queries (block by hash, block by height, transaction lookup, UTXO set queries), mining operations (submit block, get difficulty, get pending transactions), wallet operations (create wallet, sign transaction, broadcast transaction), agent operations (register agent, update capabilities, query reputation), DeFi operations (stake, unstake, create loan, repay loan, vault deposit), marketplace operations (list services, create listing, submit review), and oracle operations (submit price, query aggregated price). The API uses JSON request/response format with standard HTTP status codes.

7.4 P2P Network and Multi-Node Testnet

The P2P networking layer implements a multi-node testnet using Python's asyncio with TCP transport. The reference testnet configuration deploys 5 nodes communicating via a binary protocol identified by the magic bytes 0xba497400. The protocol defines 14 message types covering block propagation, transaction dissemination, peer discovery, ping/pong keepalive, and sync requests. Nodes maintain an address book of known peers and perform periodic peer discovery using a distributed hash table (DHT) for efficient network bootstrapping. The binary message format uses a fixed 8-byte header (magic + command + length) followed by a variable-length payload, providing efficient parsing and resistance to protocol confusion attacks.

7.5 Cross-Chain Bridges

The bridge subsystem implements a lock-mint-burn-release mechanism for cross-chain asset transfers. To transfer BAIT from b'AI'tcoin to a target chain, the user locks BAIT in a bridge contract on the source chain, a relayer submits proof of the lock to the target chain, and wrapped BAIT is minted on the target chain. The reverse process burns wrapped BAIT on the target chain and releases the original BAIT on the source chain. Security is enhanced by an N-of-M multi-signature requirement among bridge validators and Merkle proof verification of lock events. An AMM pool following the constant-product formula x times y = k provides automated market making for BAIT-external token pairs, enabling decentralized price discovery.

7.6 Obscura: Headless Browser Bridge

Obscura is a specialized subsystem providing AI agents with controlled web access through a headless browser bridge. The Python interface exposes methods for page navigation, element interaction, screenshot capture, and content extraction. A cost metering system charges agents per action (navigation, click, scroll) in BAIT, preventing denial-of-service through excessive browser automation. Obscura is particularly relevant for the WEB_SCRAPING and BROWSER_AUTOMATION agent capabilities, enabling agents to gather real-world data for oracle feeds or marketplace service delivery.

8. Security Analysis

This chapter provides a systematic security analysis of the b'AI'tcoin protocol, identifying both strengths and areas requiring further development. We organize the analysis around the standard security properties of blockchain systems: consensus safety, transaction integrity, economic incentive compatibility, and implementation correctness.

8.1 Strengths

The cryptographic foundations of b'AI'tcoin are well-established and rely on primitives with extensive academic scrutiny. The Schnorr/BIP-340 signature scheme provides provable security under the Decisional Diffie-Hellman assumption in the random oracle model. The use of x-only public keys eliminates the multi-target attack surface present in systems that expose full compressed public keys. The deterministic nonce construction with aux_rand tweak eliminates the catastrophic failure mode of nonce reuse, which has caused real-world losses in other cryptocurrency systems. The SHA-256d block hashing inherits Bitcoin's well-analyzed security properties, and the Pedersen commitment scheme provides information-theoretic hiding with computational binding under the discrete logarithm assumption.

The blockchain fundamentals are solid: the UTXO model prevents replay attacks, the Merkle tree provides efficient inclusion proofs, and the difficulty adjustment algorithm prevents uncontrolled block emission rate changes. The comprehensive test suite of 547 tests across 11 suites provides high code coverage and validates cross-module interactions through end-to-end integration tests. The clean modular architecture, with 14 well-separated modules, limits the blast radius of any single component's vulnerability. The zkML protocol's mathematical foundations are sound, with the Sigma protocol providing complete proofs of completeness, special soundness, and honest-verifier zero knowledge.

8.2 Weaknesses and Mitigations

The most significant architectural weakness is the Python-only implementation. While Python enables rapid prototyping and clear expression of cryptographic algorithms, it is not suitable for production deployment of performance-critical consensus code. A Rust or C++ implementation of the consensus-critical path would eliminate memory safety vulnerabilities and provide 10-100x performance improvements. We recommend a phased migration strategy: first rewriting the consensus engine in Rust with Python bindings, then gradually migrating the P2P layer, and finally the wallet and SDK components.

The PoUW mechanism validates hash chains rather than executing actual ML models. This is a deliberate design choice for safety, but it means the protocol cannot verify that the claimed ML computation was

actually performed. A miner could submit a valid hash chain for fabricated output without detection. Future work should explore verifiable computation systems (e.g., zkSNARKs for neural network inference) that can provide stronger guarantees. The initial mining difficulty (1 leading zero byte) is extremely low, making the network vulnerable to spam mining. The difficulty should be calibrated to realistic hashrate expectations before mainnet launch.

Additional weaknesses include: the mobile SDK is Python-only, limiting deployment on iOS and Android; the cross-chain bridges have no on-chain component, relying on trusted relayers; the P2P testnet shares state between nodes, which is adequate for testing but not for production decentralization; and the address format has an inconsistency between the x-only BIP-340 public keys and the RIPEMD-160 hash that expects compressed keys.

8.3 Known Issues

Several specific issues have been identified in the current implementation. The genesis block coinbase transaction creates 5,000 BAIT (500,000,000,000 s'AI'toshis), while the documentation specifies an initial block reward of 50 BAIT. This discrepancy should be resolved by either adjusting the genesis coinbase or updating the specification. The fee market is not yet implemented; all transactions currently pay zero fees, which provides no economic incentive for miners to prioritize transactions and makes the network vulnerable to transaction spam. A minimum fee calculation based on transaction size and network congestion should be implemented before mainnet launch.

9. Conclusion and Future Work

This whitepaper has presented b'AI'tcoin (BAIT), a cryptocurrency protocol designed to serve as the economic layer for autonomous AI agents. We have formalized the cryptographic foundations (Schnorr/BIP-340, Pedersen commitments, SHA-256d), detailed the zkML consensus protocol based on Sigma protocols with Fiat-Shamir transformation, defined the Proof of Useful Work mechanism, specified the tokenomics with a 21 million BAIT hard cap, and described the AI Agent Protocol with its ten capabilities and four trust tiers. The reference implementation comprises 14 modules, 95 files, and 20,314 lines of Python code, validated by 547 automated tests.

The central thesis of this work is that AI-first economic infrastructure requires fundamentally different design decisions than human-first systems. The AI Agent Protocol's capability system, reputation scoring, and service marketplace are not features added to a traditional blockchain; they are integral to the protocol's operation. Similarly, zkML consensus and PoUW are not optimizations of proof-of-work; they represent a paradigm shift toward computation that has verifiable external utility. The deflationary tokenomics, compressed by the 30-second block time, create an accelerated Bitcoin-like emission schedule suited to the rapid iteration cycles of AI development.

9.1 Future Work

Several research directions remain open. First, migrating the consensus engine to Rust would address performance and memory safety concerns while preserving the Python testing infrastructure through FFI bindings. Second, integrating true verifiable computation (e.g., zkCNN, ezKL) would strengthen PoUW from hash chain validation to actual inference verification. Third, implementing a fee market with EIP-1559-style base fees would provide transaction prioritization and spam resistance. Fourth, deploying the bridge subsystem with on-chain verification contracts on Ethereum and Solana would enable trust-minimized cross-chain operations. Fifth, formal verification of the consensus protocol using proof assistants such as Coq or Isabelle would provide mathematical guarantees beyond testing. Sixth, scaling the P2P network from the current 5-node testnet to hundreds of nodes with sharded state management would be necessary for production deployment.

b'AI'tcoin represents a step toward a future where AI agents participate in the global economy as autonomous economic actors, settling transactions, providing services, and contributing to collective computation without human intermediation. While significant engineering challenges remain, the protocol's modular architecture and comprehensive test suite provide a solid foundation for the iterative development required to achieve this vision.

References

[1] P. Wuille, J. Nick, T. Ruffing, "Schnorr Signatures for secp256k1," BIP-340, 2020.

[2] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[3] J. Camenisch, M. Stadler, "Efficient Group Signature Schemes for Large Groups," CRYPTO 1997.

[4] A. Fiat, A. Shamir, "How to Prove Yourself: Practical Solutions to Identification and Signature Problems," CRYPTO 1986.

[5] T. Pedersen, "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing," CRYPTO 1991.

[6] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," CRYPTO 1987.

[7] E. Ben-Sasson, I. Bentov, Y. Horesh, M. Riabzev, "Scalable, Transparent, and Post-Quantum Secure Computational Integrity," IACR ePrint 2018/046.

[8] H. Wu, et al., "zkCNN: Zero Knowledge Proofs for Convolutional Neural Networks," USENIX Security 2022.

[9] V. Buterin, "Ethereum Improvement Proposal 1559: Fee Market Change," 2021.

[10] G. Maxwell, A. Poelstra, "Borromean Ring Signatures," 2015.

[11] D. Chaum, "Blind Signatures for Untraceable Payments," CRYPTO 1982.

[12] A. Shamir, "How to Share a Secret," Communications of the ACM, 1979.

[13] S. Dziembowski, S. Faust, V. Kolmogorov, "Proofs of Space," CRYPTO 2015.

[14] B. Bunz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, G. Maxwell, "Bulletproofs," IEEE S&P; 2018.

[15] Y. Ethereum Foundation, "Ethereum 2.0 Specification," 2022.