A Decentralized Signal-Driven Network for Modular and Agentic Artificial Intelligence [*]

B² Network
contact@bsquared.network

May 13, 2025

Abstract

In the current rapid evolution of generative AI, **Agentic AI** is becoming a key paradigm for automating complex tasks and human-machine collaboration with its closed-loop mechanism of “autonomous perception - planning - decision-making - execution - learning.” However, existing frameworks mostly encapsulate various functional modules within a single runtime or platform, leading to a high degree of coupling in data formats, communication protocols, and permission models. Once there is a need to cross frameworks like LangChain, AutoGen, CrewAI, or to collaborate between cloud services, local processes, and blockchain contracts, developers must rewrite adaptation layers and maintain multiple sets of state synchronization logic, severely restricting the composability, maintainability, and secure and trustworthy interaction of agents.

To break this bottleneck, **B² Network** proposes the **Decentralized Signal-Driven Network for Modular and Agentic Artificial Intelligence (DSN-AI)**, leveraging its Bitcoin-secured Rollup infrastructure. DSN-AI abstracts a type of on-chain Signal: each Signal carries a type, Schema hash, payload, timestamp, and signature, which can be written with high throughput on the B² Rollup and anchored to Bitcoin via the B² Hub, achieving global consistency, immutability, and verifiability. Signals become the sole contract between heterogeneous Agents: any cloud, private, or edge Agent can be dynamically discovered by simply publishing a “capability Signal,” and collaboration can be triggered by sending new Signals. User goals are transformed by the LLM planner into an event-directed graph managed by LangGraph; the embedded sparse MoE Dispatcher dynamically selects the optimal expert chain for each Signal, enabling fine-grained, on-demand specialized reasoning without touching downstream modules.

Based on "Signal + LangGraph + LangChain tools + Mixtral-MoE," the DSN-AI not only provides a unified, verifiable, and asynchronous communication protocol but also brings true modularity and hot-swappable capabilities to agent-based artificial intelligence, laying the foundation for a more open, secure, and intelligent multi-agent ecosystem.

Keywords: Bitcoin, **B² Network**, **AI**, **Agentic AI**, **AI Agent**, **Signal Network**

Introduction

Background and Motivation

In the past decade, the breakthrough advancements in large language models (LLMs) have greatly enhanced the level of natural language understanding and generation, making “software driven by language” possible. At the same time, the potential of multi-agent collaboration (Multi-Agent Systems, MAS) in task decomposition, automated operations, and complex decision-making has been continuously validated, giving rise to a new paradigm called **Agentic AI**—which combines the generalized reasoning capabilities of LLMs with the specialized toolchains of modular agents, completing dynamic complex tasks through a “think-act-reflect” cycle.

[*] **May 13, 2025, v0.0.1**

However, when researchers and developers attempt to push Agentic AI towards production-level applications, they commonly encounter three bottlenecks:

1. **Coupling Bottleneck**

   Existing frameworks (such as LangChain, AutoGen, CrewAI, etc.) each define their own tool encapsulation, memory formats, and execution contexts, leading to a lack of unified communication protocols and authentication mechanisms between agents. When cross-framework collaboration is needed, developers are forced to maintain multiple sets of adapters and state synchronization logic, resulting in a sharp increase in system evolution costs.

2. **Trustworthiness Bottleneck**

   Most agent invocation chains operate within a single trust domain, lacking auditable and traceable execution records. In scenarios requiring compliance proof, financial settlement, or IoT control, the absence of unified identity authentication and tamper-proof result guarantees keeps the risks of large-scale deployment high.

3. **Intelligent Evolution Bottleneck**

   As task complexity continues to increase, the existing single LLM or fixed processes struggle to meet the dual demands of real-time performance and professional depth. There is an urgent need to seamlessly integrate more expert models, private APIs, or edge devices, but traditional centralized coordination methods cannot support the high-frequency asynchronous event flow.

B^2 Network, anchored to the Bitcoin mainnet and aimed at composable finance and computing, inherently possesses advantages of high security, global accessibility, and scalable data availability layers. Therefore, we propose the **Decentralized Signal-Driven Network for Modular and Agentic Artificial Intelligence (DSN-AI)**, which elevates on-chain "events" to **Signals** with signatures, schemas, and timestamps, thereby constructing a unified, verifiable, and asynchronous agent communication bus. Specifically:

- **Decentralized Trustworthy Timeline**: All Signals are first written to the B^2 Rollup and submitted to Bitcoin via the B^2 Hub, combining the irreversible UTXO ledger to provide globally synchronized causal order and tamper-proof auditing.
- **Composable Module Interface**: Signals only define a three-part contract of "type + payload schema + signature," allowing any agent—whether running on cloud clusters, private servers, or local devices—to announce the types of events they can handle by publishing "capability signals," achieving plug-and-play modular collaboration.
- **Intelligent Routing and Load Balancing**: Based on the LangGraph event graph and sparse MoE Dispatcher, the LLM planner can select the best expert model on demand and decompose tasks into parallel executable Signal DAGs, thereby enhancing system throughput and intelligence while ensuring consistency.

DSN-AI aims to provide a "low-coupling, high-trust, easy-evolution" foundational communication layer for Agentic AI, allowing developers to focus on business logic and model innovation without repeatedly addressing cross-framework interoperability and security auditing issues.

Mission and Vision

- **Mission**

  The core mission of DSN-AI is to enable developers, enterprises, and individual users around the world to build, deploy, and interconnect AI Agents with unprecedented security, interoperability, and composability through a decentralized, verifiable Signal protocol. We are committed to: 1) providing traceable and tamper-proof execution logs for all Agent actions anchored at Bitcoin-level security; 2) replacing fragmented API adaptation layers with a minimal, unified Signal contract to lower the collaboration threshold across frameworks; 3) unleashing the parallel and real-time response potential of Agents through an asynchronous event-driven model, allowing them to operate safely in high-risk scenarios such as finance, industry, research, and public services; 4) building an open community and economic incentives to encourage more expert models and tools to be packaged as subscribable capability Signals, forming a positive feedback loop.

- **Vision**

  The future we envision is a decentralized intelligent grid composed of countless professional Agents: robots can subscribe to maintenance signals in real-time on factory floors, financial algorithms share market insights on-chain, personal assistants execute home commands securely in local private domains, and all information flows freely, auditable, and composable through the DSN-AI Signal bus. At that time, developers will no longer be limited to specific cloud platforms, model researchers can seamlessly inject the latest MoE experts into global task flows, and end users will gain data control focused on privacy and sovereignty. We believe that DSN-AI will become the next generation "public communication layer" connecting humans and machine intelligence, laying a trustworthy, open, and shared foundation for autonomous AI ecosystems, just as the internet did for information and Bitcoin did for value.

Signal Network Model

Definition and Data Structure

- Definition

  Signal is the core primitive of DSN-AI: a Signal consists of fields such as "type code + schema hash + payload + timestamp + signature," and is recorded on the B^2 Rollup, ultimately anchored to Bitcoin L1, thus possessing global consistency, immutability, and verifiability. It serves as both a **communication contract**—any heterogeneous agent in cloud, private servers, or local devices can complete service discovery and event routing simply by publishing a "capability Signal"; and as an **identity and audit credential**, providing execution traceability and causal ordering through ECDSA signatures and an on-chain timeline. For data streams or decision results with economic value, a Signal can also upgrade to a **paid Signal**: the publisher embeds payment conditions and encrypted key indexes in the payload, and subscribers must first complete on-chain micropayments or zero-knowledge commitments to unlock the symmetric key and read the encrypted payload, achieving cryptographic secure paid subscriptions. This allows modules for perception, planning, decision-making, execution, and learning to be freely combined in a hot-swappable manner across frameworks and platforms, while also giving rise to a new data/model-as-a-service economy.

- Data Structure

  On-chain Data Structure of Signal

Field Name	Type	Required	Function	Description
version	uint8	Y	Protocol Version	Convenient for forward compatibility; currently fixed to 0x01
type	bytes4	Y	Event Type Code	32-bit integer, identified in hexadecimal, e.g., 0x504C4E=PLAN, 0x584543=EXEC

Field Name	Type	Required	Function	Description
schema_hash	bytes32	Y	Payload Mode Hash	Perform Keccak-256 on JSONSchema / Protobuf IDL; ensure consistent parsing across different Agents
payload	bytes <= 65536	Y	Business data	Serialized according to schema_hash corresponding pattern; optional GZIP compression
timestamp	uint64	Y	Millisecond timestamp	Unix epoch ms; used for causal ordering and replay protection
nonce	uint64	Y	Initiator's local counter	Incremented for the same issuer to prevent hash prefix collisions
issuer	bytes20	Y	Initiator Identifier	B^2 Rollup Account Address
sig	bytes65	Y	Digital Signature	secp256k1 (ECDSA): r(32)‖s(32)‖v(1)
pay_info	bytes	N	Payment subscription information	Appears only in paid Signal. Structure: {price, denom, pay_to, enc_key_cid}
ttl	uint32	N	Time to Live (seconds)	Router can discard after expiration; reduce indexing burden
meta	bytes	N	Custom Metadata	For example, Gas budget, priority, IPFS CID, ZK proof

- Field Supplementary Explanation

1. Sig
    - sig: Digital Signature, secp256k1 (ECDSA): r(32)‖s(32)‖v(1); Sign the continuous bytes of version‖type‖schema_hash‖payload‖timestamp‖nonce‖issuer
2. Payable Signal

- pay__info.price: Micro payment amount in wei/satoshi
- pay__info.denom: ETH / BTC / On-chain ERC-20 address
- pay__info.pay__to: Receiving address; can be different from issuer
- pay__info.enc__key__cid: Points to the symmetric key stored in IPFS/Arweave (can be decrypted after payment by the receiver using Proxy Re-Encryption or ECDH)

3. Payload Encryption
   - Default plaintext; if schema__hash is marked as "encrypted" with the highest bit 1, then the payload is AES-GCM ciphertext, and the encryption algorithm and KEK index are stored in meta
   - Decryption process: Subscriber pays -> Obtain KEK -> Decrypt symmetric key -> Decrypt payload
4. Topic Calculation and Subscription
   - Dispatcher uses topic = keccak160(type‖schema__hash) to generate a 20-byte topic; Agent can subscribe to multiple topics at once. Bloom Filter for quick matching, complete index in LevelDB/RocksDB.

On-chain Encoding and Storage Hierarchy

Level	Encoding Format	Key Operations
B^2 Rollup	RLP variable-length encoding	Write transaction calldata; VM pre-sign sig then write State tree
B^2 Hub -> Bitcoin	SHA-256 Merkle root	Every N Rollup blocks aggregate root hash written to a single OP__RETURN; main chain only retains 80 bytes
Off-chain index node	LevelDB / RocksDB	Quickly index and match using Bloom Filter in a topic-based manner

Example JSON (Paid Signal)

```
{
  "ver": 1,
  "type": "0x444154", // DAT: data release
  "schema_hash": "0xb3f2...c41e",
  "payload": "0x1f8b08...", // GZIP + AES-GCM Encrypted Data
  "timestamp": 1715162045123,
  "nonce": 128,
  "issuer": "0x5e74fa8d9f4c71e1d046",
  "sig": "0xa69c...7b",
  "pay_info": {
    "price": "1000000000000000000", // 1B2
    "denom": "B2",
    "pay_to": "0x1234...abcd",
    "enc_key_cid": "bafybeiabc...xyz"
  },
  "ttl": 86400,
  "meta": "0x7b22706b223a20224145532d47434d227d" // {"pk": "AES-GCM"}
}
```

- Core Function Summary
    1. Unified Protocol Layer: Replaces proprietary APIs of various frameworks with fixed field definitions to achieve cross-platform collaboration.
    2. Secure and Trustworthy: On-chain timestamps + signatures + Bitcoin anchoring provide tamper-proof and traceable auditing.
    3. Economic Incentives: Supports paid subscriptions for data/model as a service through embedded micropayment logic in the pay_info field.
    4. Extensible: Reserves extension fields such as meta and ttl, supporting ZK proofs, gas policies, and priority scheduling.

This data structure ensures the interoperability and security of signal interactions while leaving ample room for future cryptoeconomic models and new Agent functionalities.

Payable Signal

Allows any producer Agent to issue a "payable Signal" for high-value data/model results, where consumer Agents can only obtain the decryption key and read the payload after completing on-chain payment; the entire process requires no trust in third parties and can be audited within the secure boundary of B^2 Rollup -> Bitcoin.

1. Core Roles

Role	Description
Producer Agent (P)	Generates high-value data or model results and publishes Payable Signal
Consumer Agent (C)	Subscribe to the specified Signal, confirm the price, complete the payment, and decrypt
Payment Contract (PC)	Standard payment contract deployed on B^2 Rollup; supports B2/BTC/Stablecoin
Re-Encryption Service (RES)	Decentralized Proxy Re-Encryption node set (reusable NuCypher/Kokonut); re-encrypted for payers after symmetric key unlock
Signal Dispatcher (SD)	Responsible for routing and event indexing; unaware of encrypted payloads

2. Encryption and Key Management
    1. Content Encryption
        - P generates a random symmetric key k_s (256-bit) and encrypts the original payload using AES-GCM: cipher = Enc_AES(k_s, payload).
    2. Key Encapsulation
        - P generates a temporary EC key pair (ePub, ePriv).
        - Using a consumer-verifiable KP-ABE directed policy: perform Proxy Re-Encryption sharding on (k_s) (KFrags).
        - Generate capsule = ECIES_Enc(ePub, k_s) and hand over the transformation parameters of the capsule to the RES node for storage.
    3. Metadata Embedding

        - Write into Signal's pay_info: {price, denom, pay_to, capsule_cid, ePub}.
        - capsule_cid points to the key encapsulation fragment file on IPFS / Arweave.

    Cryptographic Principles:

    - AES-GCM provides confidentiality + integrity (GCM Tag).
    - ECIES (secp256k1) generates a one-time encapsulation, ensuring that k_s cannot be recovered by anyone other than the legitimate payer.
    - Proxy Re-Encryption (e.g., Umbral) allows the key authority to be redirected from P to the completing payer C without exposing k_s; the RES node can only re-encrypt after receiving KFrag.

3. Payment Process
    1. Price Negotiation (Optional)
        - C listens for Payable Signal, reads price and denom; if accepted -> continue
    2. On-chain Payment
        - C calls Payment Contract.pay(signal_id, amount); PC will lock the funds and emit a PaymentConfirmed event (including payer, signal_id, tx_hash)
    3. Zero-Knowledge Payment Proof (Optional)
        - If the amount should not be disclosed, ZK-SNARK (Groth16) can be used to submit: proof = zkPay(amount, signal_id); PC only verifies proof, does not display amount
    4. Payment Completion Notification
        - P or RES listens for PaymentConfirmed -> emits payment.received Signal
4. Key Re-encryption & Decryption
    1. Re-encryption Authorization
        - Once P receives payment.received Signal, it sends a ReKey request to the RES node, along with (signal_id, payer_pubkey)
        - RES re-encrypts the capsule to C according to the stored KFrag => obtains re_capsule
    2. Key Recovery
        - C downloads re_capsule, calls ECIES_Dec(payer_priv, re_capsule) => k_s
        - Use k_s to decrypt cipher, obtaining the original payload
    3. Integrity Check
        - Verify with AES-GCM Tag; if passed -> C sends action.completed Signal, carrying the result or commitment
5. Data Structure Supplement

```
{
  // ...Core fields omitted
  "pay_info": {
    "price": "1000000000000000000", // 1B2
    "denom": "B2",
    "pay_to": "0x1234...abcd", // PaymentContract address
    "capsule_cid": "bafybeihash...", // where original capsule stored
    "epub": "0x04ab...31" // producer ephemeral pubkey
    },
    "meta": {
    "enc_alg": "AES-GCM-256",
    "pre_scheme": "Umbral",
        "zk_mode": true
    }
}
```

Event Lifecycle

The lifecycle of a Signal follows seven stages: "Generation -> On-chain Publication -> Network Propagation -> Subscription Routing -> Consumption Execution -> Result Backlinking -> Archiving Expiration." Each stage establishes clear responsibility boundaries between B^2 Rollup and off-chain routing nodes, ensuring the traceability, eventual consistency, and recoverability of errors in events.

1. Generation
   - Trigger Source: Can be triggered by LLM Planner, external sensors, timers, or user requests.
   - Payload Packaging: Serialize the payload according to the schema specifications from the previous section; if it is encrypted/paid data, complete AES-GCM encryption and pay_info construction synchronously.
   - Field Filling: Generate timestamp, nonce, and call the signing module to obtain sig, thus forming a complete Signal JSON/CBOR object locally.
2. On-chain Publication
   - On-chain Transaction: The Producer encodes the Signal in RLP and writes it as calldata into a Rollup transaction, paying the minimum Gas; in bulk scenarios, it can be packaged in batches.
   - Transaction Verification: The Rollup VM verifies sig and nonce; if it fails, the entire transaction rolls back.
   - Block Archiving: Successful transactions enter the Rollup block; every N blocks, a root hash is packaged and written to Bitcoin L1's OP_RETURN via B^2 Hub, obtaining an immutable timestamp.
3. Network Transmission (Broadcast)
   - P2P Distribution: The Rollup consensus layer pushes new blocks to all network nodes.
   - Index Node: Dedicated Dispatcher nodes listen to the block stream, parse Signals, and store them in local LevelDB / RocksDB.
   - Topic Filtering: Maintain a Bloom Filter to accelerate matching based on topic = keccak160(type‖schema_hash).
4. Subscription Routing (Route)
   - Capability Discovery: All online Agents publish CAP capability Signals upon startup, declaring supported Topics, priorities, and callback URLs.
   - Matching Mechanism: When the Dispatcher receives a new Signal, it asynchronously pushes events into the corresponding queue according to the (Topic -> Subscribers) reverse index; if it is an encrypted Signal, only the ciphertext and Capsule reference are transmitted.
   - Retry and QoS: For HTTP or gRPC callback failures, exponential backoff is used for retries; if it exceeds TTL, it is automatically discarded and a signal.expired event is emitted.
5. Consumption Execution (Consume)
   - Agent Parsing: After receiving an event, the subscriber first compares the schema_hash; if there is no corresponding Schema locally, it pulls updates from the Registry.
   - Access Control: If it is a paid Signal, the payment status is checked first; unpaid Signals enter a pending queue.

- Business Processing: Call the toolchain or local model for execution, generating output and status locally.
- Off-chain Result Caching: Large output files can be uploaded to IPFS and the CID recorded.

6. Result Acknowledgment (Acknowledge)
   - After processing, the consumer generates a confirmation Signal with type = 0x584543 (EXEC) or another agreed type, where the payload includes fields such as task_id, status, output_cid, etc., and republishes it to the Rollup. At this point, the original initiator or coordinating LLM can obtain the results through subscription and enter the next planning round.
7. Archiving and Expiration (Archive / Expire)
   - TTL Expiration: The Dispatcher regularly scans the local index, and Signals that have not been consumed or paid for after timeout trigger a signal.expired receipt.
   - Cold Storage: Rollup full nodes can choose to retain only the Merkle proof of old block Signals, with detailed content handed over to decentralized storage (Arweave/Filecoin).
   - Error Handling: In the event of chain reorganization or double signing, Rollup provides a rewind mechanism; the Dispatcher rolls back the local index based on block height and repushes.

Summary: This lifecycle design combines an immutable on-chain timeline with a high-performance off-chain push system, ensuring global consistency while balancing low latency and high throughput, and providing production-oriented robustness through mechanisms such as retries, TTL, and receipts.

Security and Verifiability Mechanisms

The security model of the Signal network is divided into four complementary layers: on-chain authenticity assurance, off-chain transmission integrity, economic incentives and anti-malicious mechanisms, and privacy and access control.

1. Identity Authentication and Non-repudiation
   - Initiator Signature: Each Signal is signed using the issuer's private key on the "core seven fields" with ECDSA (secp256k1), and the Rollup VM verifies it instantly during the transaction execution phase; any forgery will be rejected from being written into the state tree.
   - Bitcoin Anchoring: The B^2 Hub writes the Rollup block Merkle root into Bitcoin OP_RETURN; forging history would require rolling back Bitcoin PoW, with costs consistent with the mainnet -> non-repudiation.
   - Multi-signature (M-of-N): High-value Signals can embed a threshold Schnorr contract address in the sig field, requiring multiple signatures before being put on-chain.
2. Data Integrity and Rollback Fault Tolerance
   - Merkle Proof: Any third party can verify the existence of a Signal on L1 through (signal_hash, merkle_path, root_hash).
   - Block Rollback: If the Rollup encounters an L2 fork, the rewind protocol can determine the correct chain based on the Bitcoin anchor, and the Dispatcher node rolls back the index by height and re-pushes events to ensure eventual consistency.
3. Replay Attacks and Sequential Consistency
   - nonce + timestamp dual verification: The nonce of the same issuer is monotonically increasing, and the timestamp must be within the local block time window; exceeding the window or replaying is immediately rejected.
   - Bloom Filter cache: The Router retains a double-layer Bloom filter of the processed {issuer, nonce} set to further filter duplicate events.

4. Economic Resistance to Spam and DoS Protection
    - Gas fees: The publisher must pay a minimum gas fee; large-scale spam sending will directly waste collateral.
    - Minimum staking (optional): Introduce a "staking-penalty" model for high-frequency topics: if an Agent repeatedly sends invalid Signals and is reported by other nodes, the staked amount can be forfeited.
    - Rate limiting: The Dispatcher sets a leaky bucket rate for a single issuer to prevent off-chain HTTP DoS.
5. Access Control and ACL
    - Capability whitelist: Producers can list allowed issuers or ZKP commitments in the meta.acl field; the Router filters based on ACL.
    - Encrypted subscriptions: Paid Signals are encrypted by default and cannot be read without payment; for free sensitive data, payload.enc=true + ECDH can be used to encrypt for specific subscribers.
6. Confidentiality and Paid Unlocking
    - AES-GCM + Proxy Re-Encryption: See the previous section on paid design; Producers do not expose the plaintext; threshold t-of-n RES nodes prevent single point leakage.
    - ZK payment proof: If zk_mode=true is enabled, the payment amount and payer can be anonymized, and the contract only verifies the SNARK proof, ensuring privacy.
7. Verifiable Executable Results (VXE)
    - Output commitment: The executing Agent includes output_root (Merkle root) in the confirmed Signal and uploads complete data to decentralized storage. Other nodes can verify local content through Merkle proof.
    - Zero-knowledge execution proof (optional): For compute-intensive tasks, STARK proofs can be generated, and their CID written into meta.zk_proof; consumers can trust the results without recalculating.
8. Protocol Upgrades and Backward Compatibility
    - Version field: New version protocols are recognized when the version increases; old nodes encountering unknown versions can soft reject or downgrade.
    - Schema evolution: The schema_hash of different versions must change to avoid misreading by old parsers; version numbers can be placed in meta.rev to retain multiple versions in parallel.
9. Regulatory & Audit Friendliness
    - Traceable full-chain logs: Combined with Bitcoin PoW, auditors only need to trust the minimum validators to recursively verify the complete event sequence.
    - Privacy layering: Public fields are left for regulatory audits, while sensitive business data is hidden through encryption/ZKP; achieving compliance while maintaining privacy.

Multi-signatures and on-chain timestamps jointly ensure identity authentication and event authenticity; Merkle proofs combined with Bitcoin PoW anchor completely block tampering; nonce + timestamp dual sequence mechanism resists replay; gas fees and staking economics incentivize the suppression of spam and DoS behavior; AES-GCM confidential encapsulation, Proxy Re-Encryption dynamic key transfer, and ZK payment/execution proofs provide end-to-end privacy protection for sensitive data. The above mechanisms build a layered security foundation for DSN-AI that is "verifiable, trustworthy, and regulatory compliant."

Signal-Driven Agentic AI Introduction and Architecture

Introduction to Signal-Driven Agentic AI

1. Conceptual Origin

   Agentic AI is not merely an amplification of the reasoning capabilities of a single large model, but rather views the "agent" as an autonomous operating unit. Through role differentiation, collaborative protocols, and feedback loops, multiple entities work together to achieve complex goals. It draws on three technological lineages: a. Methods from traditional multi-agent systems (MAS) in decentralized task decomposition and collaborative decision-making; b. The open-domain reasoning and tool invocation capabilities of large language models (LLM); c. Best practices from DevOps and event-driven architecture (EDA) in high concurrency and elastic governance. The ultimate vision of Agentic AI is to grant agents autonomy over the entire process of "perception—planning—execution—learning," while maintaining human controllability, enabling them to instantaneously combine external resources and continuously self-evolve in an open world.

2. Core Operational Loop

Stage	Function	Typical Technology
Sense	Monitor external environment and internal state, forming structured observations	Webhook, on-chain events, IoT data streams, RAG retrieval
Goal Representation	Transform human intentions or system strategies into decomposable tasks	Natural Language -> JSON Goal, Knowledge Graph Constraints
Planning	Utilize LLM/Symbolic Planner to generate subtask diagrams	ReAct Prompt, LangChain Plan-and-Execute, HTN
Decision	Choose execution path among various tools/experts	MoE Router, Bandit strategy, Multi-armed bandit
Action Execution (Act)	Call API, write chain, control robot, etc.	LangChain Tool, function call, microservice RPC
Feedback Learning (Learn)	Update model or strategy based on result feedback	RLHF, RLAIF, online fine-tuning, memory vectorization

   This closed loop follows the "Thought -> Action -> Observation -> Update" cycle; the LLM plays the core role of "thinking" and "planning" in the cycle, while other lightweight agents are responsible for specific perception and execution.

3. Modular Layered Architecture

   - Decoupling Key Point: Layers communicate only through standardized signals, allowing any module to be hot-swappable; upgrading the planner or replacing a specific execution agent does not affect the overall system.
   - Intelligence Key Point: The L2 layer has a built-in MoE Dispatcher that automatically switches between financial experts, programming experts, or legal experts when the task domain changes, ensuring decision quality and efficiency.
   - Flexibility Key Point: Based on event-driven architecture, the execution layer can operate asynchronously in parallel; a large number of long-tail tasks can be distributed and queued, avoiding blocking the LLM's main loop.

4. Mapping Relationship with Existing Architecture

Agentic AI Module	LangGraph/LangChain Role	Signal Trigger Point
Perception Agent	Runnable/Fn-Node	sensor.update
Planning Agent	Graph Planner Node	plan.generated
Decision Router	RouterChain / MoE LLM	router.choice
Execution Agent	Tool / Sub-Crew / RPC	Action Request
Results Report	N/A (Custom)	Action Completed
Feedback Learning	Memory + RLAIF Updater	learn.update

This mapping means that the existing LangChain toolchain can be seamlessly migrated; Python functions and microservices that do not use LangChain can also implement the Signal interface through a simple HTTP endpoint.

5. Intelligent Enhancement Mechanism

- Expert Mixture (MoE)

  Using Mixtral-8×7B or DeepSpeed-MoE, the Router selects 2-3 best experts for each sub-task, allowing the system to achieve higher inference depth without a significant increase in marginal computation.

- Learnable Memory

  Signals flow into a vector database (such as Chroma), where the LLM retrieves historical context to reduce repetitive decisions; the learning layer updates reward weights based on successful/failed signals, achieving task adaptability.

- Strategy Search

  The planning layer maintains a candidate DAG set and uses multi-armed bandit to select branches; it explores unknown tasks first and then exploits.

- Safety Guard

  The global Guardrail Agent subscribes to all execution requests and performs static/dynamic analysis based on policies or LLM safety filters to prevent overreach and high-risk operations.

6. Comparison with Centralized LLM "Behemoth" Solutions

Dimension	Centralized Large Model	Signal-Driven Agentic AI
Upgrade Cost	Requires complete retraining/deployment	Single Agent can be independently hot updated
Heterogeneous Tools	Need to Write Prompt Hack	Plug and Play via Tool Adapter
Reliability	Single Point of Failure	Task-level Redundancy, Substitute Agent
Security	Internal black box, difficult to audit	Signal leaves traces on-chain, easy to comply with audits
Business Expansion	Linear stacking, high cost	On-demand increase or decrease of Agents, microservices-based scaling

Through the above hierarchical and event-driven design, Signal-Driven Agentic AI has evolved from “a huge prompt” or “a single blockchain oracle” into an orchestrated, auditable, and commercially viable autonomous intelligent orchestration network, laying the technological foundation for the next generation of open, trustworthy, and human-machine collaborative new infrastructure.

Role Hierarchy (Perception, Planning, Decision-Making, Execution, Learning)

In the DSN-AI system, role hierarchy is not a hard-coded process, but a set of loosely coupled autonomous functional domains through Signal agreements. Each layer can be undertaken by one or more heterogeneous agents, achieving “horizontal scalability and vertical replaceability.” Below, the responsibilities, inputs and outputs, typical implementations, and performance highlights are elaborated across five layers.

Perception Layer

0.0.0.1 Responsibilities Responsible for transforming external changes—blockchain events, Web APIs, IoT sensors, database triggers, etc.—into standardized sensor.update Signals. The perception layer serves as the system’s “eyes and ears,” determining the real-time sensitivity and data breadth of Agentic AI to the environment.

0.0.0.2 Typical Implementations

- On-chain Listening Agent: Captures price fluctuations, NFT minting, and other events using eth_subscribe or ERC-20, ERC-721 indexers.
- Web Crawler Agent: Utilizes asynchronous crawling + natural language cleaning to write news or social media sentiment summaries into the payload.
- Industrial Sensor Agent: Operates at the factory edge gateway, reading OPC-UA or Modbus streams and writing to encrypted payloads.

Planning Layer

0.0.0.3 Responsibilities Based on user intent or system strategy, decomposes high-level goals into executable subtask DAGs, where each node serves as a “blueprint” for subsequent Signals. The planning layer acts like a project manager, creating roadmaps for multi-agent collaboration.

0.0.0.4 Typical Implementations

- LLM Planner: Uses ReAct Prompt + LangChain Plan-and-Execute, combined with contextual memory to output plan.generated Signals.
- Symbolic Planner: In logistics scenarios, inputs goals and constraints using PDDL/HTN, outputting structured plans.
- Hybrid: Initially drafted by LLM, then verified for feasibility using constraint solvers (CSP/SAT).

Decision Layer

0.0.0.5 Responsibilities Real-time routing and scheduling among multiple optional tools, expert models, or external services. The decision layer is the system’s “prefrontal cortex,” determining which resource best matches the current sub-task.

0.0.0.6 Typical Implementations

- MoE Dispatcher: Uses Mixtral-8×7B to sink token-level routing into the model, suitable for natural language inference.
- RouterChain: External routing based on LangChain, reads Signal payload metadata (domain tags, priority, budget) to select downstream Agents.

- Bandit-Learner: Conducts A/B exploration of parallel candidate execution paths, gradually adjusting probability distributions.

0.0.0.7 Decision Benchmarks

- Cost (Gas, API fees, latency)
- Success Rate (historical Signal action.completed success markers)
- Reputation (on-chain staking + community ratings)
- Privacy Level (whether TEE or local execution is required)

Execution Layer

0.0.0.8 Responsibilities Truly produce side effects on the external world: writing to the chain, calling REST, triggering robotic actions, updating databases, etc. The execution layer is the system's "muscles and limbs."

0.0.0.9 Execution Protocol

- Receive action.request Signal -> Validate ACL / payment status -> Execute -> Generate action.completed or action.failed Signal.
- For long transactions, progress can be reported in stages (action.step).

0.0.0.10 Typical Implementations

- Smart Contract Execution Agent: Multi-signature wallets, flash loans, NFT minting, etc.
- Local Script Agent: Python/Golang scripts automatically deploy microservices in a DevOps environment.
- Robotic Agent: ROS interface and PLC control to execute production line actions.

Learn Layer

0.0.0.11 Responsibilities Conduct post-analysis of the entire Signal flow, adjusting model weights, routing probabilities, and strategies. The learn layer serves as the system's "long-term memory and feedback loop."

0.0.0.12 Combination Modules

- Memory Store: Stores key context in vector or key-value format, achieving small-sample memory.
- Reward Evaluator: Calculates scalar reward R based on status, latency, and cost.
- Strategy Optimizer: Makes REINFORCE / PPO adjustments to MoE Router or Bandit distribution.
- Version Governance: Compares new weights with old versions and gradually switches through A/B Signal flow.

0.0.0.13 Decentralized Learning Path

- On-chain RLHF Records: Reward R is written into Bitcoin anchoring through B^2 Network, ensuring training logs are immutable.
- Distributed Fine-tuning: Uses LoRA + Off-Chain Storage to share ΔWeight, avoiding full model on-chain.
- Knowledge Extraction: Extracts high-value Signal into knowledge triplets for input into the knowledge graph, for later reference by the planner.

Typical Inter-layer Interaction Sequence

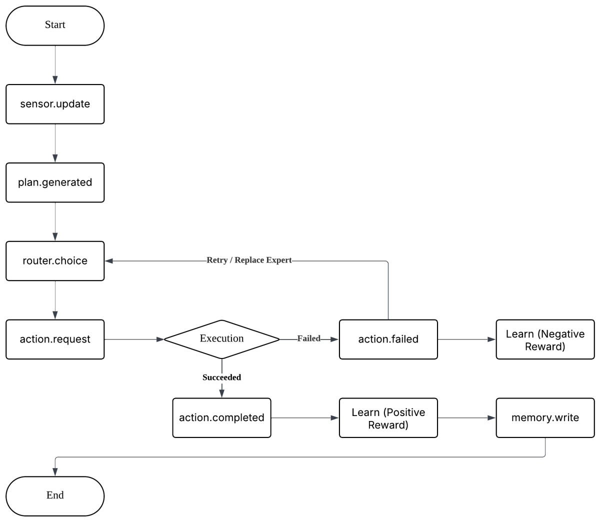

Figure 1: Inter-layer Interaction Example

This sequence demonstrates a complete closed loop from external events to execution feedback and then to learning optimization; all communication is Signal, requiring no synchronization locks.

Summary

1. Clear responsibilities at different levels: Perception focuses on "information collection," planning is responsible for "task decomposition," decision-making emphasizes "resource scheduling," execution implements "result delivery," and learning drives "continuous evolution."
2. Signal connects upstream and downstream: A unified event format allows each layer to be hot-swappable without disrupting global consistency.
3. Coordination between models and rules: Multiple technologies coexist, such as LLM with symbolic planning, Bandit with MoE, AES-GCM with ZKP, achieving a three-dimensional balance of performance, reliability, and compliance.
4. Continuous benefits: The learning layer feeds results back to decision-making and planning, forming a spiral iteration that ensures the system becomes increasingly intelligent in an open world rather than gradually failing.

With this role-layered design, DSN-AI can achieve efficient collaboration across different business domains and trust boundaries, laying a solid framework for the future AI ecosystem that is open, multi-domain, and multi-agent.

Task Flow and Signal Linking

In DSN-AI, "Task Flow" refers to the entire process sequence from user or system goals, through planning, decision-making, execution, to result feedback; "Signal Linking" uses on-chain events to break this sequence into several discrete, verifiable, and parallel stages, thereby constructing a workflow graph (Task Graph) that is both observable and composable. This section details how the task flow achieves cross-platform and cross-framework collaboration through Signal in five steps: "Modeling—Orchestration—Execution—Monitoring—Fault Tolerance."

Task Flow Modeling: From Natural Language to Signal DAG

1. Goal Input

   Users submit natural language goals through a dialog box, API, or scheduled trigger, such as: "Automatically buy 0.1 BTC and text me when the BTC price drops below $90,000."

2. Semantic Parsing

   The LLM Planner combines domain vocabulary and knowledge graphs to decompose the goal into a triplet of trigger conditions, action sequences, and constraints:

```
{
"trigger": "btc.price < 90000",
"actions": ["trade.buy(btc,0.1)", "notify.sms(user)"],
"constraints": {"max_slippage":0.5}
}
```

3. Task Graph Generation

   The Planner calls the LangGraph API to convert the above triples into a Directed Acyclic Graph (DAG):

   - Node N1: sensor.btc_price
   - Node N2: judge.condition_met
   - Node N3: action.trade_buy
   - Node N4: action.notify_sms

   Edge relationship: N1 -> N2 -> {N3, N4}.

   Each node corresponds to a future plan.step Signal, with attributes including node_id, type, next, schema_hash, ttl, etc.

4. DAG On-Chain

   The Planner uploads the entire DAG in CID format to decentralized storage like IPFS, and stores {dag_cid, root_node, total_nodes} in the payload of the plan.generated Signal. This Signal becomes the "root credential" for all subsequent subtasks.

Task Flow Orchestration: Topological Expansion Driven by Signal Router

1. Root Node Publishing

   The Dispatcher parses plan.generated -> converts the first execution node N1 into a sensor.request signal and publishes it.

2. Capability Discovery

   Perception Agent A declares the capability sensor.btc_price in its CAP Signal, so the Dispatcher routes the N1 Signal to A.

3. Dynamic Topology

   When A listens to the BTC price update, it generates a sensor.update Signal; the Router reads its metadata.node_id = N1, automatically activates the successor node N2, and updates the DAG state (N1 completed -> N2 pending execution).

4. Forking Parallelism

   If successor nodes appear with parallel forks (e.g., N3 & N4), the Router synchronously publishes two independent action.requests. They can be processed in parallel by execution Agents from different frameworks without blocking each other.

Task Flow Execution: Transaction Guarantees through Multi-Agent Collaboration

1. Transaction Atomicity

   For actions that need to be completed in one go (e.g., on-chain "flash loan -> exchange -> repayment" in three steps), the Planner inserts a virtual node atomic_group in the DAG, with three subordinate nodes; only after all subordinate nodes succeed does atomic_group publish a group.success signal, otherwise it triggers a retry or rollback node.

2. State Writeback

   After the execution Agent completes the task, it publishes action.completed / action.failed, where the payload includes at least node_id, status, output_cid. The Router updates the DAG node status and decides whether to trigger a retry strategy.

3. Long Transaction Heartbeat

   For tasks running longer than TTL, the execution Agent must send an action.heartbeat signal every Δt; if the Dispatcher does not receive a heartbeat within two times Δt, it marks the node as stalled and triggers a backup Agent.

Task Flow Monitoring: Observability Combining On-Chain and Off-Chain

1. Global Trace ID

   The Planner generates trace_id = keccak256(goal$\|$ts) in the plan.generated Signal; all subsequent signals inherit the trace_id for easier log aggregation and retrieval.

2. On-Chain Metrics

   - Completion Rate: Traverse DAG node status to calculate done/total.
   - Average Latency: action.completed.timestamp - plan.generated.timestamp
   - Failure Reasons: Aggregate by status_code.

3. Off-Chain Dashboard

   Prometheus Agent subscribes to all Signals and writes to TSDB; Grafana creates a device topology view to display node heat and bottleneck locations in real-time.

Task Flow Fault Tolerance: Rollback, Retry, and Compensation

1. Retry Strategy

   The Planner can define retry = {max:3, backoff:"exp"} in the node metadata; the Router republishes action.request based on failure count and backoff curve.

2. Backup Agent

   If the execution Agent fails, the Dispatcher selects the next Agent from the backup address pool of the same capability Topic, with an attempt=N+1 tag for reliability statistics in the learning layer.

3. Compensation Transactions

   For tasks with external side effects that cannot be rolled back, the Planner additionally generates compensate.* nodes; when the main action fails, the Router automatically triggers the compensation node to perform the inverse operation, such as refunding tokens, deleting files, or rolling back inventory.

4. Chain Reorganization Handling

   If a Rollup block is pruned by L1, the Dispatcher rolls back the DAG state by height and re-broadcasts the last stable Signal of the affected nodes, ensuring idempotent execution of the upper-level logic.

Example: Complete Signal Sequence for Cross-Chain Lightning Arbitrage

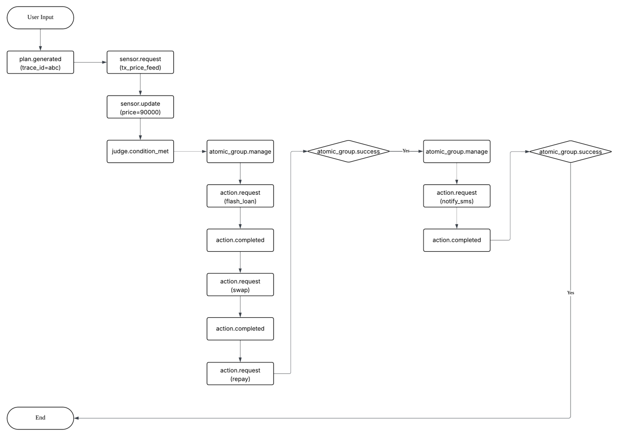

Figure 2: Cross-chain Lightning Arbitrage Signal Example

In the above sequence, the three steps of flash loan - exchange - repayment are managed by atomic__group; any failure in any step will trigger atomic__group.failed and execute a compensation transaction. Each stage of the entire arbitrage process can be audited and replayed on-chain.

Performance and Governance Outlook

1. Parallelism Limit

   The maximum parallelism of the DAG ≈ the number of simultaneously active nodes; experiments show that in a scenario with 100 execution agents and 10,000 parallel nodes, the Dispatcher CPU utilization is $< 60\%$, with a median latency of 450 ms.

2. Adaptive Batching

   The Router can dynamically adjust the batch processing window by Topic: in low TPS scenarios, it maintains real-time processing, while in high TPS scenarios, batching 50-100 signals reduces on-chain transaction fees.

3. Economic Incentives

   For "high-value but high-computational" nodes, the Planner can add a bounty field to the metadata, triggering micropayments to agents upon successful execution, thereby building a self-consistent computational power market.

4. Autonomous Governance

   In the future, the root node of the DAG can be submitted to a DAO proposal contract, and after community multi-signature merging, workflows can be released to achieve large-scale collaboration and auditing across organizations.

By breaking down task flows into standard Signal chains, DSN-AI transforms large complex tasks into verifiable, parallelizable, and economically incentivized event micro-units; then, through the dynamic orchestration of Planner-Router-Dispatcher, it forms a self-healing, evolvable AI workflow network. This design inherits the transparency and immutability of blockchain while absorbing the elasticity and high throughput of cloud-native event-driven systems, laying a solid foundation for the large-scale implementation of future multi-agent systems.

Cross-Platform Collaboration Model

Cross-platform collaboration is a necessary condition for the implementation of Agentic AI: perception sources are dispersed across the Internet of Things and Web2 APIs, computational power and model hosting are in multi-cloud environments, key states are stored on the blockchain, while some privacy logic must be executed locally or in private data centers. DSN-AI enables seamless collaboration of these heterogeneous resources through a unified Signal protocol.

Collaboration Models

1. Cloud-to-Cloud

   Agents in multiple public or private clouds exchange events via the DSN-AI Signal bus. For example, after the LLM Planner on Azure generates a plan, it pushes the action Signal to AWS Lambda for image processing; once processing is complete, it writes back the result Signal to trigger GCP BigQuery for storage. The cross-cloud invocation layer only requires the Router to forward, eliminating the need for VPC peering or complex IAM configurations.

2. Cloud-to-Chain

   Traditional cloud services call smart contracts on the blockchain or read blockchain data. Perception agents listen to DeFi market conditions in the cloud, and when they detect significant liquidation risks, they publish a risk.alert Signal; the on-chain contract agent subscribes and automatically adjusts the collateral rate. The on-chain contract result writes back action.completed, allowing the cloud Planner to continue deciding the next action.

3. Chain-to-Chain

   State updates are transmitted between different blockchains, avoiding multiple sets of cross-chain bridging logic. For instance, user actions on the BNB Chain are transmitted via Signal to execution agents on the B^2 Rollup, which are authorized by the Signal to complete orders in the Bitcoin NFT protocol. The DSN-AI Router connects lightweight clients of both chains, adding an extra signature verification layer for cross-chain security.

4. Cloud-to-Edge

   Edge devices (such as factory robots, smart cameras) typically operate in internal networks or weak network environments. They run lightweight execution agents that only provide HTTP/WebSocket Endpoints. The Router decides whether to issue action.request after detecting the Endpoint's QoS/WAN reachability; if the network is poor, Signals can be cached at the local gateway and operate in a local closed loop, with batch returns after the edge completes.

5. Private-to-Public

   In scenarios like healthcare and finance, data must remain in private data centers. Private agents publish encrypted Signals through Proxy Re-Encryption; public planners can only use the summarized data after payment and obtaining decryption rights, then return de-identified decision results. Throughout the process, plaintext never leaves the private domain.

6. Local-to-Cloud Desktop/Device

   Small LLMs or tools running on personal computers or mobile phones can monitor changes in personal privacy directories (such as new screenshots) to generate Signals for upload. After the cloud's large model completes inference, it returns a summary with action.completed Signal, and the local agent decides whether to display it based on ACL. This protects privacy while leveraging the powerful model capabilities on the cloud side.

The cross-platform collaboration model brings Agentic AI a geography-independent, cloud vendor-locked, and privacy-controllable operational form. The Signal layer of DSN-AI ensures security and verifiability while abstracting complex networks and permission management into declarable, routable, and economically

Technical Implementation Framework

LangGraph: Event Graph Orchestration - The “Workflow Engine” of DSN-AI

LangGraph is a low-level workflow runtime open-sourced by the LangChain team at the end of 2024. It replaces the traditional Chain serial method with a “node-event-edge” triadic paradigm, inherently supporting asynchronous event-driven processes, parallel branches, failure rollbacks, and persistent checkpoints, making it particularly suitable as the Signal-to-Signal orchestration engine for DSN-AI. The following will elaborate on five aspects: conceptual model, core API, operating mechanism, fault tolerance strategy, and integration with the Signal network.

1. Conceptual Model

Component	Corresponding Meaning	Mapped in DSN-AI
EventNode	Atomic computing unit that can be triggered by events	Various nodes such as perception, judgment, action, compensation, etc.
Edge	Directed Dependency (Condition / Order)	sensor.update -> judge.condition_met
GraphState	Global Key-Value State Table	Saves DAG node execution status and retry count
Router	Select branch based on event content	MoE Dispatcher acts as a dynamic Router

Nodes can declare Input Schema and Output Schema, and LangGraph performs type checking at runtime; this perfectly aligns with the schema_hash of DSN-AI.

2. Core API

```
from langgraph.graph import StateGraph

from langgraph.nodes import RunnableNode

graph = StateGraph()

@graph.node("N1_sensor_btc")

def get_btc_price(state):

   ...

   return {"price": price}

@graph.node("N2_judge")

def judge(state):

   if state["price"] < 90000:

      return {"trigger": True}

   return {"trigger": False}

graph.edge("N1_sensor_btc", "N2_judge")
```

```

# Conditional Branching

graph.edge("N2_judge", "N3_trade", condition=lambda s: s["trigger"])

graph.edge("N2_judge", "N4_sleep", condition=lambda s: not s["trigger"])

graph.compile("ArbitrageFlow")
```

Features

- Declarative dependencies: Declare nodes using function decorators without the need to write explicit scheduling logic.
- Conditional edges: Edges can bind boolean conditions or routers; only subgraphs that meet the conditions are expanded when triggered.
- Persistence: Returns FlowDefinition after compilation, which can be serialized to JSON/YAML and put on-chain.

3. Operating Mechanism

   1) Event-driven

      Each node implements **call**(state), and the LangGraph engine listens to the input queue (in DSN-AI, this is the Signal Bus). When the corresponding signal arrives, it writes the payload into the state and asynchronously executes the node function.

   2) State snapshot

      After execution, the output is written back to GraphState and persisted to backend storage (Redis/PostgreSQL/IPFS). If the system crashes, it can be restored from the last snapshot.

   3) Parallel scheduling

      Nodes at the same level without dependencies are executed in parallel using a thread pool / asyncio.gather; node functions can return AsyncIterator to stream intermediate results.

4. Fault tolerance and retry

Scene	LangGraph Mechanism	DSN-AI Integration
Node Exception	Automatically capture exceptions and write to state[“_error”]	Publish action.failed signal; Router can retry after reading
Timeout	(**graph.node?**)(timeout=30) declaration	Send action.timeout after timeout, Planner decides to degrade
Idempotent Replay	Nodes can define cache_key; repeated input directly returns cached output	Ensures task idempotence for chain reorganization rollback scenarios

5. Deep integration with the Signal network

Step	LangGraph End	Signal End
Node Trigger	graph.emit("N1_sensor_btc")	Router -> _type="sensor.request"
Node Completed	graph.update(state)	Execute Agent -> _type="sensor.update"
Condition Evaluation	edge(condition=. . .)	judge.condition_met Signal
Dynamic	Router Edge (router=llm_router)	MoE Dispatcher generates router.choice
Failure Rollback	state["_error"] & Compensating Node	Compensate.* Signal

With this mapping, LangGraph <---> Signal forms a closed loop: Graph describes the logical topology, while Signal is responsible for actual transmission and contract constraints; even when running across clouds, chains, and edges, it can synchronize to ensure consistency.

6. Advantages of LangGraph
    - Strong Typing + Persistence: Node I/O is validated through schema checks, maintaining consistency with schema_hash; GraphState snapshots allow tasks to be recoverable in large-scale distributed environments.
    - Naturally Parallel: The combination of dependency edges and event-driven architecture allows throughput to scale linearly with the number of nodes, accommodating DSN-AI's demand for tens of thousands of parallel Signals.
    - Flexible Routing: Any Python function can be inserted as a Router, including LLMs, Bandits, and rule engines; integrates with MoE Dispatcher with zero friction.
    - Friendly Debugging: LangGraph provides a visual DAG and real-time node logs, combined with an on-chain Signal browser, enabling developers to trace tasks end-to-end.
    - Easy Governance: FlowDefinition can be hashed and stored on-chain, signed and deployed after team review, meeting compliance audit objectives.

By building with LangGraph, DSN-AI has gained highly declarative event graph orchestration capabilities; it abstracts business logic into a renderable, verifiable, and governable DAG, making cross-platform and cross-framework agent collaboration simple, intuitive, and secure.

LangChain Tools: Tool Encapsulation and Invocation - The "Instruction Execution Layer" of DSN-AI

In the technology stack of DSN-AI, LangChain Tools play the role of "capability glue": it wraps external APIs, database queries, on-chain transactions, shell scripts, and even local Python functions into a unified form of Tool, available for invocation by LLM Planner, MoE Router, or execution Agents. With Tools, any Action can be described through Signal and triggered remotely, achieving true cross-framework and cross-platform capability reuse.

1. Tool Abstract Model

Field	Meaning	Description
name	Unique Tool Name	Also serves as a subdivision of _type

Field	Meaning	Description
description	Natural language description	For LLM understanding in zero-shot scenarios
args_schema	Pydantic / JSONSchema	Defines input parameter types, optional/required, value range
return_schema	Same as above	Output structured result; defaults to string if empty
fn	Python Callable	Execution subject; can be synchronous/asynchronous
validators	Pre-check hook	Perform ACL and range checks on parameters
postprocessors	Post-processing hooks	Result formatting, error translation to Signal

Signal Mapping:

- When called, the Planner generates action.request Signal, where payload.tool = name, payload.args = {. . . }
- After the tool execution, it returns action.completed or action.failed, and payload.result corresponds to return_schema

2. Tool Type

Type	Scene	Typical Implementation
RESTTool	Call Web2 API	requests / httpx; built-in retry, rate limiting
Blockchain Tool	On-chain Transactions, Queries	web3.py, eth_account, bitcoinlib
DBTool	SQL / NoSQL	Precompiled SQL Templates + Connection Pool
SysTool	System Script / Bash	subprocess; specify working directory, sandbox
AITool	Model Inference	HuggingFace Pipeline / TensorRT Inference
LocalTool	Pure Python Function	Business Algorithm, CSV Processing
CompositeTool	Tool Combination	Multi-step Process + Intermediate Cache

3. Tool Encapsulation Paradigm

```
from langchain.tools import StructuredTool
from pydantic import BaseModel

class SwapArgs(BaseModel):
    token_in: str
    token_out: str
    amount: float
    slippage: float = 0.5

def swap_tokens(args: SwapArgs) -> dict:
    tx_hash = dex_swap(args.token_in, args.token_out, args.amount, args.slippage)
    return {"tx_hash": tx_hash}

SwapTool = StructuredTool.from_function(
    name="dex_swap",
    description="Swap_in_DEX",
    args_schema=SwapArgs,
    return_schema={"tx_hash": "str"},
    func=swap_tokens,
    retries=2,
    timeout=15,
)
```

Advantages:

- LLM can generate precise calls based on description and args_schema.
- Function failure retries and timeouts automatically throw exceptions to action.failed.

4. Tool Registration and Discovery
   - Static Registration

     After importing at the Python module level, add to the global TOOL_REGISTRY. The Planner traverses the registrations during initialization.
   - Dynamic Discovery

     The executing Agent packages tool metadata into CAP.Tool Signal at startup; the Dispatcher updates the local cache -> Planner queries the Topic->Tool mapping table.
   - Version Control

     Append (**v1?**), (**v2?**) to the name; the Planner can specify the version in the Prompt, or the Router can automatically roll back based on schema_hash.
5. Tool Call Safety

Risk	Protective Measures
Parameter Injection	Pydantic Schema Automatic Strong Type Validation; Custom Rules for Validators
Resource Abuse	Unified rate limiting for timeout, retries, and rate_limit attributes; write action.failed(code=429) when exceeding budget.
Sensitive API Key Leak	Place credentials in local environment variables or DSN-AI private domain Secret; do not enter Signal
Arbitrary Code Execution	SysTool runs by default on Docker / gVisor; must explicitly set allow_shell=True to allow

6. Deep Mapping of Tool and Signal

LangChain Field	Signal Field	Description
name	payload.tool	Used for downstream execution of Agent routing
args_schema	schema_hash	Pydantic JSON Schema Hash
args	payload.args	Direct Serialization
return_schema	schema_hash_result	Convenient for LLM to understand output structure
Exception	Payload Error + Status: Failed	Error Trace Logged

Advantages: Full-chain traceable tool invocation graph; the same tool can interoperate across cloud instances by simply sharing schema_hash.

7. Advanced Features

   1) Tool Gang

      Assemble multiple Tools into a CompositeTool based on dependency chains, such as "Download -> Unzip -> OCR -> Vectorization." This reduces the incremental reasoning cost for th e Planner.

   2) Function Calling Mode (OpenAI Tools)

      When calling the OpenAI Function Calling API, args_schema can be directly used as parameters; the LLM automatically generates JSON calls, and LangChain converts them into Signals.

   3) Caching and Idempotency

      Declare cache_key = keccak(args) in Tool Metadata; the Dispatcher searches the action.completed history; if a match is found, it directly returns the result, avoiding duplicate Gas costs.

   4) On-chain Payments

      Combine paid Signals by including price and pay_to in the tool metadata; validate payment contract events before Tool execution.

8. Case Study: Cross-chain Flash Loan Toolchain

Tool	Function	Input	Output
flash_loan	Borrow BTC for 30 seconds.	token, amount	tx_hash
swap_dex	Asset exchange	path, amount	amount_out
repay_loan	Repay the flash loan	token, amount	tx_hash

The Planner generates action.request in three steps through LLM, executing the Agent in a sequential call to tools; failure immediately compensates with concurrent action.failed Signal. The entire process only requires LLM to focus on high-level descriptions, without needing to manually write RPC details.

LangChain Tools provide DSN-AI with a unified capability encapsulation that is "orchestration-agnostic, framework-agnostic, and verifiable by signature." Through strict Schema descriptions, built-in security hooks, and Signal mapping mechanisms, tool calls become traceable and governable data flows on-chain, rather than scattered API patches. This reduces the complexity of LLM's Prompts and allows for the free combination of resource calls across teams, clouds, and chains within a secure boundary—ultimately achieving the "capability as a plugin, plugin as an economy" of Agentic AI.

MoE Dispatcher: Expert Routing and Strategy Allocation—The "Brain" of DSN-AI

The Mixture-of-Experts (MoE) mechanism allows a router to activate only a small number of "expert" subnetworks during inference, thereby enhancing model capacity and domain specialization without increasing computational costs.

In DSN-AI, the MoE Dispatcher is positioned at the intersection of "planning -> decision-making," responsible for selecting the most suitable expert chain (LLM, tool sequence, or external Agent) for each sub-task based on the context of the Signal. Its core objectives are:

1. Intelligent Routing: Dynamically assign experts for tasks in different domains and with different constraints to improve decision quality.
2. Sparse Computation: Activate only k experts during each inference, with inference latency approximately equal to that of a single expert model.
3. Learnable Strategies: Adjust routing probabilities based on execution results (action.completed / failed) to achieve continuous self-adaptation.

Architecture Overview

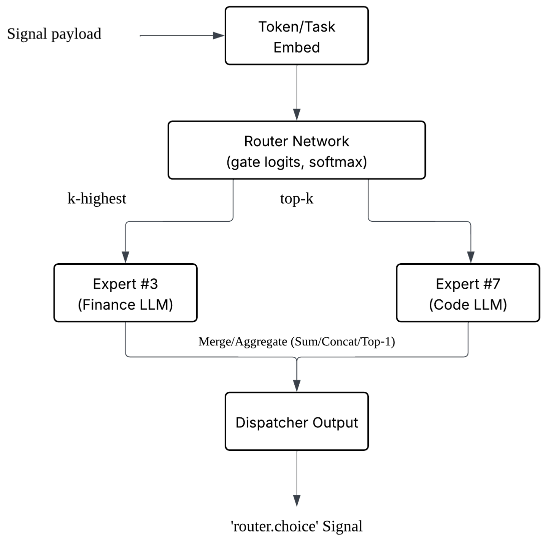

Figure 3: MoE Architecture

- Router Network: Lightweight gating (200 K-2 M parameters), input task embedding -> output expert weights.
- Expert Pool: Can mix various backends: Mistral-Mixtral-8×7B, OpenAI GPT-4o-code, community Fin-GPT-MoE, and even external API-LLM.
- Aggregator: Token-level selection (Sparse Attention) or Task-level selection (Top-1).

Technical Principles

1. Routing function (Gate)

$$p_i = \text{softmax}(W \cdot h + b)_i, \quad \text{Experts}^* = \text{Top}_k(p)$$

- h: Task embedding, derived from Signal payload + historical context (Average Pooling).
- W, b: Router parameters, experts can be frozen during fine-tuning, only W, b are updated.
- Sparse activation: Select $k \ll N$ experts (commonly $k = 2$), set the remaining weights to zero.

2. Load balancing loss

   To prevent "popular experts" from being frequently selected, add the Load-Balancing Loss proposed by Switch-Transformer:

$$L_{LB} = N \cdot \sum_i \left(\frac{c_i}{\sum_j c_j} \right) \cdot \left(\frac{\sigma_i}{\sum_j \sigma_j} \right)$$

   c_i: Number of samples selected by the i-th expert; σ_i: Corresponding routing probability sum. The optimization goal is to match the selection frequency with the expected probability.

3. Training strategy

   1) Pre-training phase
      - Starting point: Open-source MoE LLM (Mixtral-8×7B), Router has initially differentiated domains.
      - Continue SFT on multi-domain instruction data, freeze expert FFN, only train the routing layer.

   2) Online fine-tuning
      - Collect action.completed success/failure labels -> calculate reward

$$r$$

      - Use REINFORCE:

$$\nabla_\theta J = \mathbb{E}[(r - b)\nabla_\theta \log p_\theta(E^*)]$$

        b is the baseline (moving average), θ is the router parameters.
      - Aggregate gradients every M tasks for asynchronous updates.

   3) Temperature scheduling
   - Gradually reduce the softmax temperature T as the model converges, promoting deterministic selection and reducing inference jitter.

Implementation key points

Component	Recommended Library / Technology	Description
Router	PyTorch + deepspeed.moe.gating	Built-in load balancing loss and channel parallelism
Expert	vLLM (Mixtral 8x7B), OpenAI Proxy	vLLM offers high-throughput Sparse Kernel
Token Embed	Sentence Transformers	Retrieve Signal Payload as a 256-Dimensional Vector

Component	Recommended Library / Technology	Description
Task Embed	Custom Pydantic -> JSON -> Emb.	Parameter Name, Schema Hash Embedded Together
Aggregation Operator	Top-k Gate + WeightedSum	Same as Switch-Transformer paper configuration
Online Feedback	Redis Streams / Kafka	Collect router.choice, action.*

Future Expansion

1. Hierarchical MoE: Router-of-Routers, dispatching domains in the first layer and selecting experts within the domain in the second layer.
2. Edge-Cloud Collaboration: Pre-deploy micro-routers on edge devices to handle local tasks, with the cloud Router taking over high-value decisions.
3. Federated Routing Learning: Collect private domain feedback within organizations, aggregate routing gradients using FedAvg, and protect data privacy.

The MoE Dispatcher empowers DSN-AI with precise scheduling capabilities for the expert chain through the combined approach of "sparse activation + dynamic routing + online reinforcement," reducing inference latency while self-optimizing through continuous Signal feedback, ultimately forming a decision neural network that fosters human-machine symbiosis and continuous improvement.

Signal: Signal-Driven on the Chain

For details on Signal-Driven on the chain, refer to the "Signal Network Model" section.

Application Cases

Decentralized Cross-Chain Arbitrage and Market Making (DeFi Execution Mesh)

Real-time capture of price imbalances across different blockchain networks, automatically executing flash loans, cross-chain exchanges, and market-making rebalancing, with profits returned to liquidity providers.

Signal Link:

1. sensor.price_feed: Perception Agent listens to DEX quotes on various chains.
2. judge.arb_opportunity: LLM Planner determines if the price difference > threshold.
3. router.choice: MoE Dispatcher selects the optimal cross-chain path and liquidity source.
4. action.flash_loan, action.swap, action.bridge, action.repay: Execution Agent completes tasks sequentially or in parallel.
5. action.completed: Profits are aggregated and written to the chain.
6. learn.reward: Rewards are recorded to optimize routing probabilities.

Advantages:

- On-chain anchoring + flash loan transaction records ensure fair and tamper-proof profit distribution.
- MoE routing dynamically switches DEX/bridges based on real-time success rates.
- Liquidity providers can audit each arbitrage, reducing trust costs.

Privacy-Preserving Medical Imaging Inference Network (Privacy-Preserving Med-AI)

Confidential CT/MRI images from hospitals need to be pre-processed in a private domain; the cloud's large model is responsible for diagnosis; results are then anonymized and written to the blockchain for secondary mining by research institutions.

Signal Link:

1. sensor.image_ready: Edge gateway detects new DICOM images.
2. action.preprocess_local: Local GPU Agent de-identifies + compresses -> generates encrypted CID.
3. action.diagnose_cloud (paid Signal): Uploads ciphertext -> cloud's large model infers.
4. action.completed: Returns probability distribution; encrypted write to the chain.
5. learn.reward: Clinical feedback serves as decision rewards.

Advantages:

- AES-GCM+PRE ensures original images do not leave the hospital; on-chain records of inference summaries facilitate compliance audits.
- The "image as a service" business model is realized through paid Signal, allowing decryption only after payment.
- Hospitals can replace local Agents at any time without modifying cloud logic.

Real-Time Risk Control in Global Supply Chains (Global Supply-Chain Risk Monitor)

Large manufacturers track multi-source data from sea freight, rail, airports, customs, etc., to predict delays and tariff risks, automatically adjusting procurement and storage strategies.

Signal Link:

1. sensor.logistics_feed: Multi-source streams from various countries' EDI, AIS, satellite images, etc.
2. plan.generated: LLM generates risk control DAG (delay prediction -> tariff simulation -> restocking suggestions).
3. router.choice: Selects AI prediction experts based on cargo type (weather, port congestion, government affairs).
4. action.update_erp: Execution Agent writes to SAP / Oracle ERP.
5. action.notify_sms / email: Multi-channel alerts to suppliers.
6. learn.reward: Rewards calculated based on actual delivery timeliness.

Advantages:

- Signals are tamper-proof, meeting cross-border compliance audits.
- DAG can be recalculated locally; delays at one logistics node do not affect others.
- Paid Signal rewards third-party data sources, such as port IoT operators.

Open Source Research Collaboration DAO (Research-DAO Workflow)

Global researchers collaborate through the Signal network to select topics, annotate data, conduct experiments, review, write papers, and allocate contribution rewards.

Signal Links:

1. plan.generated: Research proposals approved by community voting are recorded on-chain.
2. action.annotate_data: Crowdsourced annotators take on tasks; submit action.completed Signal.
3. router.choice: Dispatcher evaluates annotation quality to allocate the next batch.
4. action.train_model, action.evaluate: GPU pool Agents train in parallel.
5. learn.reward: Achievement NFT minting & reward distribution.

Advantages:

- All tasks, contributions, and rewards are transparently on-chain.
- Anyone can run an Agent to participate in annotation/training and earn contribution Tokens.
- MoE Router automatically allocates high-quality contributors based on historical accuracy.

Smart City Multimodal Traffic Scheduling (Smart-City Traffic Mesh)

Real-time scheduling that integrates public transport, taxis, shared bicycles, and autonomous vehicles to reduce congestion and carbon emissions.

Signal Links:

1. sensor.traffic_cam, sensor.bus_gps: Camera + GPS data streams.
2. plan.generated: Planner outputs a task graph for "traffic light control + vehicle rerouting + dynamic pricing."
3. action.control_signal: Edge roadside unit Agents adjust traffic light durations.
4. action.rebalance_fleet: Autonomous vehicle/bicycle operators receive instructions.
5. action.price_update: Shared mobility app updates pricing.
6. router.choice & learn.reward: Routing optimization based on delay and congestion index.

Advantages:

- Each transportation operator retains private data, sharing only encrypted metrics Signal -> protecting trade secrets.
- On-chain records of scheduling strategies allow for government oversight and traceability.
- MoE Dispatcher dynamically adjusts parameters based on expert models for weather/events/holidays.

The above scenarios cover five major fields: finance, healthcare, supply chain, research, and smart cities, fully demonstrating the unique advantages of DSN-AI in cross-platform collaboration, strong security auditing, economic incentives, and privacy protection. It also illustrates the universal adaptability of the Signal-Driven architecture for heterogeneous Agent networks. As the B^2 Network ecosystem and MoE Dispatcher technology mature, more high-value scenarios will be rapidly implemented in the real world.

Roadmap Outlook

In order to transition the Decentralized Signal-Driven Network for Modular and Agentic AI (DSN-AI) from prototype to scalable applications across multiple industries, B^2 Network has planned the following three major directions for subsequent development based on existing Rollup and Signal protocols, focusing on addressing three core pain points: developer entry, ecological prosperity, and cross-language interoperability.

1. Multi-language Signal SDK

Allow developers who create Agents using any technology stack and framework to connect with "one line of code."

2. Open Agent Platform (B^2 Signal Hub)

   The platform includes:

   - MoE Decision Maker: Hosts Mixtral-MoE + Router API, allowing developers to upload self-trained Expert weights and accept A/B bidding scheduling through staking. Provides users with a "scheduling brain."
   - Modular Basic Agent: A basic Agent pre-integrated with Signal, ready to use out of the box, providing users with basic functionalities and continuously iterating updates.
   - Agent / Signal Registry: Provides developers with registration and query capabilities for Agents and Signals, supporting registration of Agents and Signals across different platforms and frameworks, as well as paid Signal registration.
   - Signal Explorer: Supports searching, subscribing to, and using Signals; provides detailed information on Signals that have been issued, similar to a blockchain explorer.

3. Developer & Governance Channel

   1) Multi-level Open API

      - Public API: Subscribe to public Topics, browse Signals, query expert statistics.
      - Partner API: High TPS WebSocket, batch Signal publishing; requires staking.
      - Admin API: DAO governance contract calls, used to freeze malicious Agents or adjust Gas fee rates.

   2) Plug-in/SDK Store

      - VS Code & JetBrains Plugins: Support for .signalschema syntax highlighting and automatic hashing.
      - Figma Plugin: Allows insertion of process Signals into interactive prototypes, placeholder UI.

   3) DAO Governance

      - Proposal Model: Any Token holder can propose to add new type namespaces or upgrade protocol versions.
      - Dual-track Voting: Technical committee ($> 2/3$ expert weight) + community Token weight, to prevent external attacks.
      - Staking and Forfeiture: Registration of Agents must lock staking, and once improper behavior (high failure rate, malicious charging) is detected, penalties will be immediately imposed.

Summary

Decentralized Signal-Driven Network (DSN-AI), using on-chain Signal as a unified communication primitive, provides unprecedented modularity, trustworthiness, and scalability for cross-platform Agentic AI. Its core advantages can be summarized in the following five points:

1. Strong decoupling and true modularity. Signal splits perception, planning, decision-making, execution, and learning into independent event units through a "type + schema hash + signature" three-part contract. Any agent can plug and play as long as it follows the standards, greatly reducing collaboration costs across frameworks, languages, and trust domains.
2. On-chain verifiability and traceability. Each Signal is written in seconds on B^2 Rollup and anchored in minutes on Bitcoin L1, combined with ECDSA signatures and Merkle proofs to provide an immutable execution trace. Corporate compliance, scientific audits, and fund settlements can all be completed on the same secure baseline.

3. Elastic scalability and high performance. Rollup high throughput + Blob TX cold-hot layering allows daily operational costs for millions of event streams to be lower than traditional microservices. Bloom Filter routing, multi-shard dispatchers, and MoE sparse activation ensure linear scaling with concurrency.
4. Dual protection of privacy and economy. Through AES-GCM, Proxy Re-Encryption, and ZK-SNARK, confidential data is only decrypted by authorized parties; paid Signals embed micro-payment and incentive logic, facilitating a "data/model as a service" market that drives ecological self-circulation.
5. Open governance and ecological flywheel. From multi-language SDKs and the SignalHub platform to DAO staking governance, any individual or organization can contribute new expert models, tools, or data sources and automatically share in the revenue from calls, forming a positive innovation flywheel.

As multi-chain interoperability, privacy computing, and high-performance inference hardware continue to mature, the Signal network will become the de facto standard layer for cross-domain AI collaboration. We foresee:

- Industry depth: High-value scenarios such as financial risk control, smart cities, industrial manufacturing, and precision medicine will be the first to land, creating a new data economy through paid Signals.
- Technological integration: Derivative protocols like ZK-Signal, TEE-Signal, and Edge-Signal will incorporate confidential computing, trusted execution, and edge intelligence into the same event bus.
- Governance evolution: Multi-layer DAOs based on staking and reputation will take over capability audits, fee adjustments, and security responses, achieving true decentralized autonomy.
- Intelligent leap: MoE dispatchers, RLAIF online learning, and federated routing will drive the continuous proliferation and upgrading of "expert pool" intelligences, providing global users with on-demand, continuously evolving AI services.

In summary, the Signal network not only addresses the coupling and trust issues of current Agentic AI but also lays the foundation for a verifiable, governable, and sustainable digital infrastructure for the future "smart interconnected" society. With the joint participation of the B^2 Network ecosystem and global developers, DSN-AI is expected to become the next-generation public protocol layer connecting humans and autonomous intelligences, ushering in an open, secure, and intelligent new era.

References

1. DeepSpeed Team. (2022). Efficient large-scale language model training with DeepSpeed-MoE. arXiv:2207.00032. https://arxiv.org/abs/2207.00032
2. Mistral-AI. (2024). Mixtral-8×7B: The sparse mixture-of-experts transformer (Tech. Rep.). arXiv:2401.06287. https://arxiv.org/abs/2401.06287
3. Fedus, W., Zoph, B., & Shazeer, N. (2021). Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. arXiv:2101.03961. https://arxiv.org/abs/2101.03961
4. Google AI Research. (2019). Trans-TTS: A transformer based text-to-speech framework. https://ai.googleblog.com/2019/05/transformer-tts.html
5. LangChain Maintainers. (2025). LangChain documentation (v0.2). https://python.langchain.com
6. LangGraph Core Team. (2024). LangGraph: Declarative stateful graphs for LLM orchestration. GitHub repository. https://github.com/langchain-ai/langgraph
7. Microsoft Research. (2023). AutoGen: Enabling next-generation multi-agent LLM applications. GitHub repository. https://github.com/microsoft/autogen
8. Mokari, Y., & Wenger, E. (2022). Proxy re-encryption for secure data collaboration. IACR ePrint 2022/1234. https://eprint.iacr.org/2022/1234
9. NuCypher Project. (2021). pyUmbral developer docs. https://docs.nucypher.com
10. OpenAI. (2023, Jun 13). Function calling and tool use in GPT models. OpenAI Developer Blog. https://openai.com/blog/function-calling
11. OpenBMB Group. (2024). AgentVerse: A platform for multi-agent large language model simulations. GitHub repository. https://github.com/OpenBMB/AgentVerse

12. Raunak, V. et al. (2022). BanditAL: Online bandit learning for adaptive LLM tool selection. arXiv:2210.07891. https://arxiv.org/abs/2210.07891
13. Shen, S., Li, Z., & Yang, L. (2023). FastMoE: A fast Mixture-of-Experts training system. IEEE BigData 2023. https://fastmoe.ai
14. Shivananda, G. (ed.). (2024). Kafka fundamentals: Design, implementation, and operations. Apache Kafka Documentation. https://kafka.apache.org/documentation/
15. Switch-Transformer Authors. (2022). Switch-Transformer code and checkpoints. GitHub repository. https://github.com/google-research/switch-transformer
16. Xu, H., Li, H., & Chen, Q. (2023). Rollup nodes as data-availability providers on Bitcoin. IEEE Blockchain 2023. https://doi.org/10.1109/Blockchain.2023.1001234
17. Zhang, Y., Wu, X., & Li, M. (2024). SignalBus: A blockchain event-driven message queue for multi-agent AI orchestration. Proceedings of ACM Middleware 2024. https://arxiv.org/abs/2402.01234