Persistent AI

Powerful Orchestration Engine for Telegram Native Use

February 2026

Table of Contents

Summary

PersistentAI is a Telegram-native agentic intent engine - what OpenClaw pioneered for personal AI assistants, extended with deterministic orchestration, native micropayment settlement (x402), and on-chain auditability (TetraChain). If OpenClaw made Telegram a place where AI acts on your behalf, PersistentAI makes it a place where AI agents transact, compose, and settle - with every step metered, every creator paid, and every execution formally verified.

As enterprise analyst Matt Slotnick recently argued, the most valuable systems of the coming decade will shift from managing records of what things are to orchestrating intent around what needs to happen - from objects to objectives. PersistentAI is the Telegram-native embodiment of this shift: a system of intent for the 950M-user messaging economy, where user objectives are decomposed into deterministic execution graphs, every settlement, inference and execution being settled on-chain.

TL;DR

PersistentAI provides a **deterministic orchestration layer** that bridges the gap between probabilistic foundation models and mission-critical applications. Built on a **Haskell core**, the platform enforces **formal verification** and **exactly-once execution semantics** via **DBOS**, eliminating the hallucination and non-determinism risks inherent in generative AI. This architecture enables the deployment of AI agents in high-stakes sectors like finance and identity, where structural reliability is a non-negotiable requirement.

Technical Architecture & UIO

The platform utilizes Telegram as its primary context substrate, resolving UX fragmentation through **User Intent Orchestration (UIO)**. Using a bidirectional transformation primitive, PersistentAI disaggregates Mini Apps (TMAs) and data sources into atomic **Nodes** with strictly defined fields and input/output ports. These nodes are recomposed into **unified execution graphs** managed by an **Orchestration Agent** that decodes multi-layer user intents and coordinates specialized task delegation across the network.

The Agentic Economy & Settlement Layer

The **Agentic Settlement Layer (ASL)**, powered by **TetraChain**, serves as the high-throughput payment rail for the ecosystem. Every node operates as an independent economic agent via an **x402 port**, facilitating autonomous, per-inference micropayments. This framework enables the **tokenization of inference**, where AI logic and datasets are represented as **NFTs** with automated revenue splits settled instantaneously on-chain between providers, creators, and owners.

Verification and Interface Standards

To maintain execution integrity, the **AG-UI protocol** renders dynamic, context-aware interface components as deterministic wrappers around node outputs, preventing bug propagation within the chat thread. Privacy is maintained via **Trusted Execution Environments (TEE)** for private inference, ensuring sensitive data remains shielded from model providers. All operations are immutably logged on **TetraChain**, providing a formally verified audit trail that guarantees absolute alignment between user intent and system execution.

Key Takeaways

1. Contemporary AI stacks lack a deterministic orchestration layer between probabilistic foundation models and end-user applications, resulting in fragmented UX that is simultaneously ill-suited to high-risk domains.

2. **PersistentAI** provides this layer: a Haskell-core, formally verified, deterministic execution environment for AI agents - essential where hallucination and non-determinism are unacceptable.

3. Our core value proposition is the provision of a rich substrate for **User Intent Orchestration (UIO).** UIO refers to the on-going shift from reactive, fragmented navigation to proactive, intent-driven fulfillment of user intents via autonomous agents.

4. **Telegram** is the optimal environment for UIO: a superapp rich in contextual surface, yet fragmented across hundreds of bots and mini-apps (TMAs). PersistentAI resolves this via **bidirectional transformation - any TMA, app, or data source becomes a PersistentAI node** (autonomous, economically self-sovereign agent); and vice versa - any agentic flow can be deployed as a TMA.

5. Nodes are atomic, each with defined I/O ports and an **x402 payment port.** They compose into **unified execution graphs:** a single user intent triggers deterministic traversal across requisite nodes.

6. The **AG-UI** protocol renders context-aware interface components (product cards, comparisons, confirmations) directly in the chat thread - deterministic wrappers around node outputs, not probabilistic bug-prone 'vibe-code'.

7. **DBOS** guarantees durable, exactly-once execution for mission-critical flows (irreversible transactions, financial operations, critical fact checks, etc.).

8. **Every node is an independent economic agent.** Nodes charge per-inference micropayments via x402; inference tokenisation enables price discovery at micro-scale, creating a liquid marketplace for intelligence. Each inference triggers automated, instant payment splits across model providers, node creators and/or owners, data providers (e.g. RAG maintainers), and protocol treasury - all settled on high-throughput TetraChain.

9. **TetraChain** powers the **Agentic Settlement Layer (ASL):** a high-throughput, low-latency payment rail for Agent-to-Agent, Human-to-Agent, and Agent-to-Human transactions, enabling trustless interaction with programmable money and DeFi.

10. **Every interaction and execution step is immutably recorded on TetraChain**, producing a formally verified audit trail that guarantees all actions remain strictly aligned with original user intent.

11. **Trusted Execution Environments (TEE)** enable private inference on sensitive data without exposure to LLM providers; on-premise deployment satisfies more nuanced privacy requirements, such as those required for regulatory compliance.

12. When it comes to UIO, however, the ultimate abstraction is the **Unified Chat Interface:** a single, continuous, **persistent**, context-aware thread. An Orchestrator (chat-specific assistant agent) decomposes intents, delegates to specialized Agents (indexers, settlement rails, DeFi primitives), and renders dynamic UI via AG-UI.

13. **Instantiation: UIO in practice.** A user types a complex, multi-constraint request into Telegram - e.g., purchase of a technical product with specific components AND automatic cashback conversion to Bitcoin. PersistentAI decomposes this into an execution graph of dozens of autonomous nodes: inventory scrapers, compatibility verifiers, pricing oracles, payment authorisers, cashback detectors, YouTube scrapers, and crypto-swap engines. All execute in parallel, settle micropayments across each economic actor, and render an interactive, auditable receipt - **all within the same chat thread, in under 90 seconds.**

14. Telegram's unique density of integrated contexts - public channels, private chats, group discussions, and a rich ecosystem of transactional mini-apps-constitutes a semantic factory floor where user intents can be expressed, contextualised, and acted upon. No other platform possesses this combinatorial richness of social graph, payment rails, and lightweight application distribution.

15. The user never leaves Telegram, never pastes an address, never manually sweeps a reward. The interface is alive; the economy is embedded; the intent is formally satisfied.

16. **Conclusion**: When **Telegram** is instrumented as the execution substrate for User Intent Orchestration, the messenger transcends its chat origins to become a **Universal Intent Interface**. Every inference is metered, every creator is instantly paid, every step is immutably recorded on TetraChain, and every intent is formally proven satisfied. The old paradigm of navigating fifty browser tabs, ten apps, and three wallets collapses into a single continuous conversation with persistent context.

    This, however, is only possible at scale and for a wide range of use-cases **if and only if the orchestration is deterministic**. With PersistentAI it is.

Overview

PersistentAI Enables Formal AI Orchestration

PersistentAI, a **deterministic orchestration layer for AI Agents, serves as a** critical infrastructure for maintaining control, context, and auditability of the underlying core AI infrastructure. In laymen terms, it serves as the central nervous system integrating any ecosystem built with or around AI-powered primitives, semantic, programmatic, user-generated or algorithmic, probabilistic or deterministic.

It is the layer in the AI stack that sits between the foundational models like GPT or Claude and the app layer that end-users (and Agents) interact with. But as we demonstrate below, the execution layer can also integrate the apps into a single UX creating a de-facto higher level abstraction in the AI stack. We will elaborate upon this below.

The platform enforces formal verification through its Haskell core and Deterministic Execution - a critical feature in high-risk execution environments, such as finance-related applications or apps making use of sensitive user data. The Haskell core functions as the logic compiler and formal verification engine, which generates execution-ready tasks for the DBOS-managed worker nodes. As for the former, PersistentAI ensures reliability in AI-powered financial operations mitigating the risks inherent in frequently hallucinating probabilistic AI. This architecture transforms context-rich platforms into factories for bespoke intelligence, integrating among other things:

Summary

1. **Agent Runtime & Orchestration:** PersistentAI provides the visual workflow builder and execution engine for coordinating multi-Agent logic, tool integration, and complex, event-driven pipelines.

2. **Settlement Infrastructure:** Powered by TetraChain (TON Layer 2), Agentic Settlement Layer (ASL) provides the high-throughput, low-latency payment rail for Agent-to-Agent (A2A), Human-to-Agent (H2A) and Agent-to-Human (A2H) transactions, supporting private identity, KYC delegation, Agent registries and compliance via zero-knowledge proofs.

PersistentAI's integration of a high-performance settlement layer with its orchestration engine is a key differentiator. It provides the deterministic payment rail required for Agents to interact trustlessly with programmable money and DeFi primitives.

This enables the seamless, automated financial compositions-from simple Agentic purchases to complex structured products-that represent the next frontier of financial services.

- **Payment Rails and Settlement:** Utilization of x402 as a primary internet-native payment layer, enabling high-frequency, low-latency settlement for transactions with or between agents.
- **Permission Orchestration:** Implementation of the AP2 (Agent Payments Protocol) for managing mandates, ensuring every financial action is cryptographically signed and placed on-chain and stays within user-defined bounds.
- **Indexing Infrastructure:** Integration with The Graph and Alchemy to maintain real-time awareness of blockchain states through high-fidelity data feeds.
- **Asset Support:** Out-of-the-box functionality for transferring ERC-tokens and native assets across all EVM-compatible networks.
- **Chain-Agnostic Operating System:** A comprehensive architectural approach that provides a unified financial operating system for AI across multiple blockchain ecosystems.

3. **DeFi & Traditional Protocol Mesh:** Connectivity to DeFi primitives and legacy financial gateways, enabling hybrid product composition. Structured products sourcing liquidity from DeFi enable unparalleled diversity of financial products, opening new revenue streams for banks, neobanks and wallets.
4. **Identity & Compliance Layer:** A unified system managing decentralized identity (DID), user consent, and regulatory adherence. This allows for compliant DeFi operations and delegating financial operations to AI Agents.
5. **Secure computation:** Deployed on-premise and utilizing TEE for inference, PersistentAI enables security of user data allowing for integration of sensitive datasets originating from 3rd party platforms. Private inference will become a significant compliance requirement in most jurisdictions and also a growing demand from privacy-mindful users.

→ However, the core value proposition of PersistentAI lies elsewhere. Enter **User Intent Orchestration (UIO)**[1].

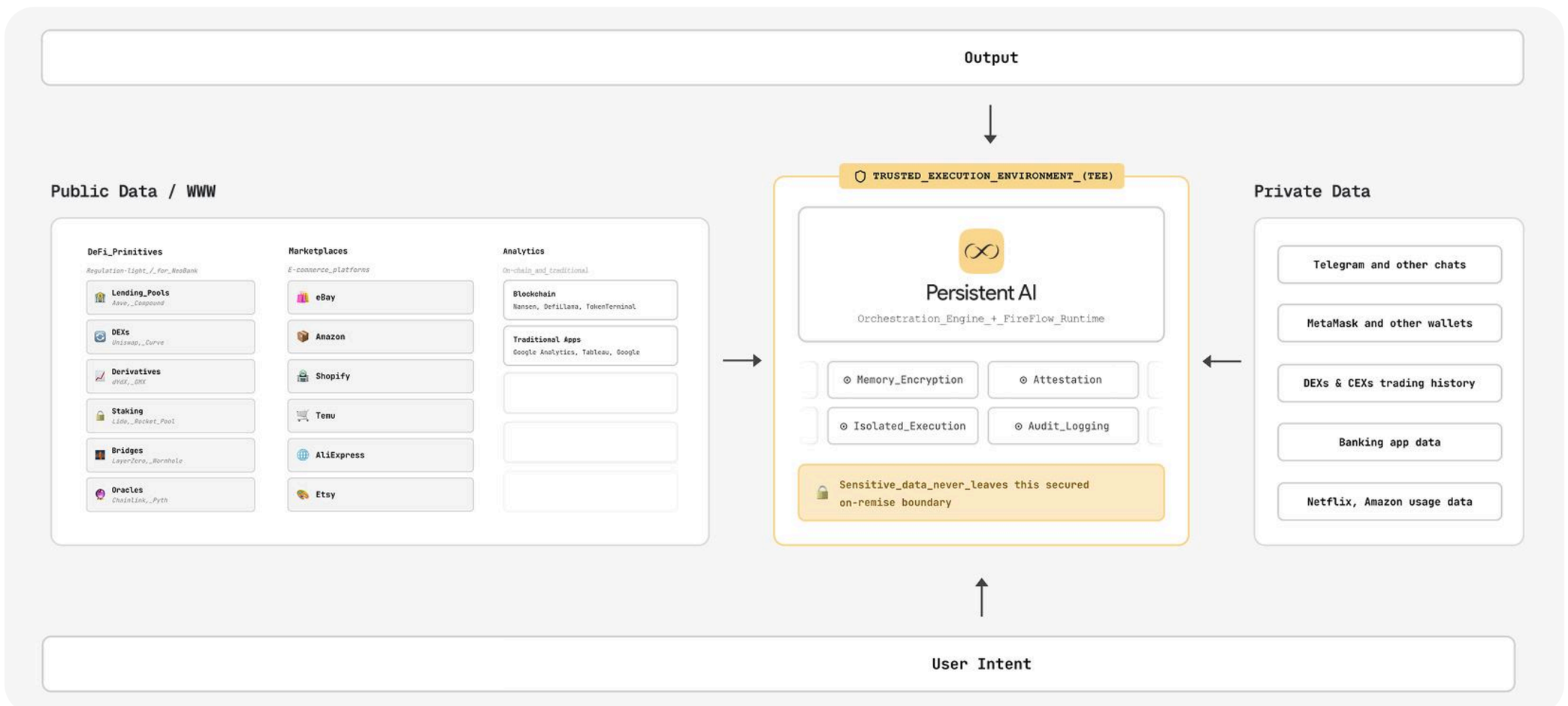

Fig1: User Intent Orchestration (UIO)

[1] Refer to Appendix 2 for a detailed discussion about UIO.

Benefits of User Intent Orchestration (UIO)

- More seamless, intent-driven experience across the entire Telegram ecosystem - no app switching, no manual aggregation.
- More powerful user interactions in one chat multiple TMAs and data sources are coordinated in a single thread.
- Secure execution via TEE - combining on-chain state, private user data, and third-party feeds into a single deterministic inference.
- Wider range of attainable actions from DeFi execution to agentic commerce, all without leaving the chat.

Fig2: Benefits of User Intent Orchestration

6. **UIO - Solving User Intents:** Front-end AI Agents (those augmenting end-user experience) are evolving into autonomous co-pilots, managing among other things personal finances and executing transactions, which shifts the banking relationship to delegated agency and risks disintermediation by third-party platforms. Already today AI systems can drastically improve user experience across a wide range of applications starting from personal finance, research, personalized medical consulting, lending, and, notably, e-commerce and intent-driven banking.

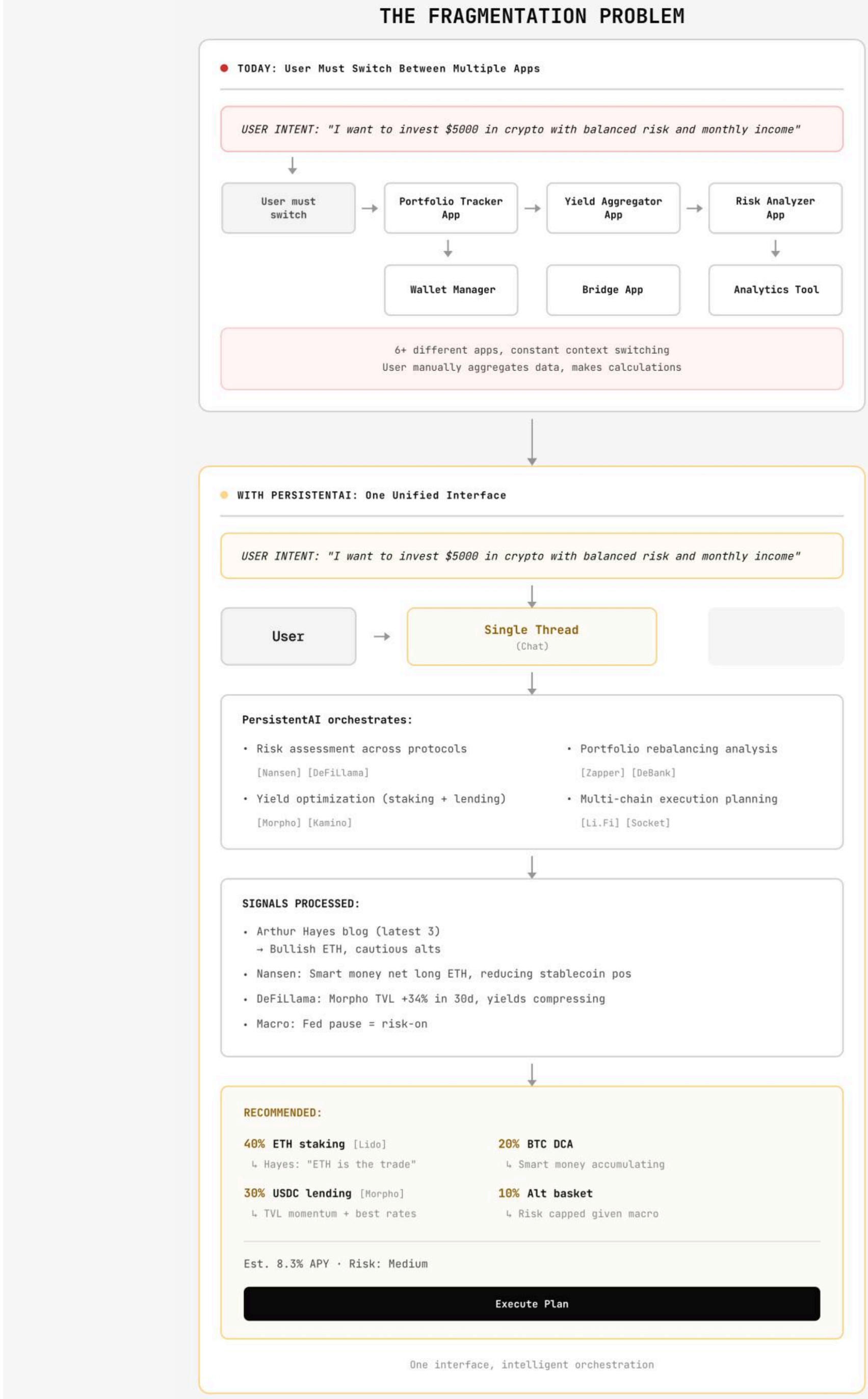

Fig3: The Fragmentation Problem

- **The Emergence of Intents as Dominant UX Paradigm:** As AI becomes more and more relevant in each individual's life, it becomes increasingly important to provide users with not only comfortable UX (we call this term AX - Agentic Experience in the context of AI) but also ways to participate in the emerging economy.
- **Telegram is the Perfect Intent Substrate:** Telegram has already become not just a messenger but the true "Everything App" where users chat, transact, get information, etc. It therefore possesses one of the richest substrates for encapsulating user intents - in other words, many different everyday user scenarios can be navigated through using Telegram's existing functionality.
- **Telegram Does not yet Work with User Intents:** Telegram is the optimal substrate: a superapp rich in contextual surface, yet fragmented across hundreds of bots and TMAs. PersistentAI resolves this via bidirectional transformation-any TMA, app, or data source becomes a PersistentAI node (autonomous, economically self-sovereign agent); any agentic flow built in the platform deploys as a TMA.
- **The Process of Fulfilling One's Intent is Costly:** For a user to express their intent they have to navigate across this diverse set of sources, or switch to a different app altogether, i.e. a process of expressing an intent is spread across multiple interfaces inside (and outside) of Telegram, making the experience fragmented, inconvenient and frequently error prone.
- **Development of TMAs is Also a Costly Process:** For those not possessing sufficient development expertise TMAs are hard to build. This restricts many enthusiasts who might have interesting ideas but just cannot wrap them into the working product.
- **Enter PersistentAI:** PersistentAI addresses these hurdles by allowing developers and users to transform traditional TMAs into PersistentAI nodes. These nodes function as autonomous AI Agents that operate within a deterministic orchestration layer. This process is bidirectional: any existing TMA can be turned into nodes to join the ecosystem, and any AI Agent flow created within the platform can be deployed as a functional TMA.

→ **PersistentAI integrates existing TMAs, public groups, 3rd party apps, and even user private data into a deterministic AI orchestration layer to proactively solve user intents.**

Node-Based Intelligence: Bidirectional Transformation

PersistentAI's core innovation is treating every computational capability as a **node** within a deterministic orchestration layer. These nodes function as modular building blocks that can be composed into complex AI Agents. The system supports bidirectional transformation:

TMA → PersistentAI Node: Any existing TMA can be converted into a Node to become part of the PersistentAI ecosystem. This process involves:

1. Wrapping the TMA's API endpoints as node interfaces
2. Defining input and output ports that other nodes can connect to
3. Registering the node in the PersistentAI catalog
4. Setting pricing for node usage (if the creator chooses to monetize)

This means existing TMA developers can enhance their applications with AI capabilities without rebuilding from scratch. A weather TMA becomes a weather node. A payment TMA becomes a payment node. The functionality is preserved but becomes composable.

PersistentAI Agent → Deployed TMA: Conversely, any AI Agent flow created within PersistentAI can be deployed as a functional TMA. The platform automatically:

1. Generates a web interface that adheres to Telegram's WebApp standards
2. Handles authentication and user context management
3. Provides default UI components or renders custom AG-UI components
4. Manages state persistence across user sessions
5. Deploys the TMA to PersistentAI's infrastructure (or the developer's preferred hosting)

This bidirectional capability creates network effects: each new node increases the value of the entire ecosystem, and each deployed TMA can potentially become a node for other developers to build upon.

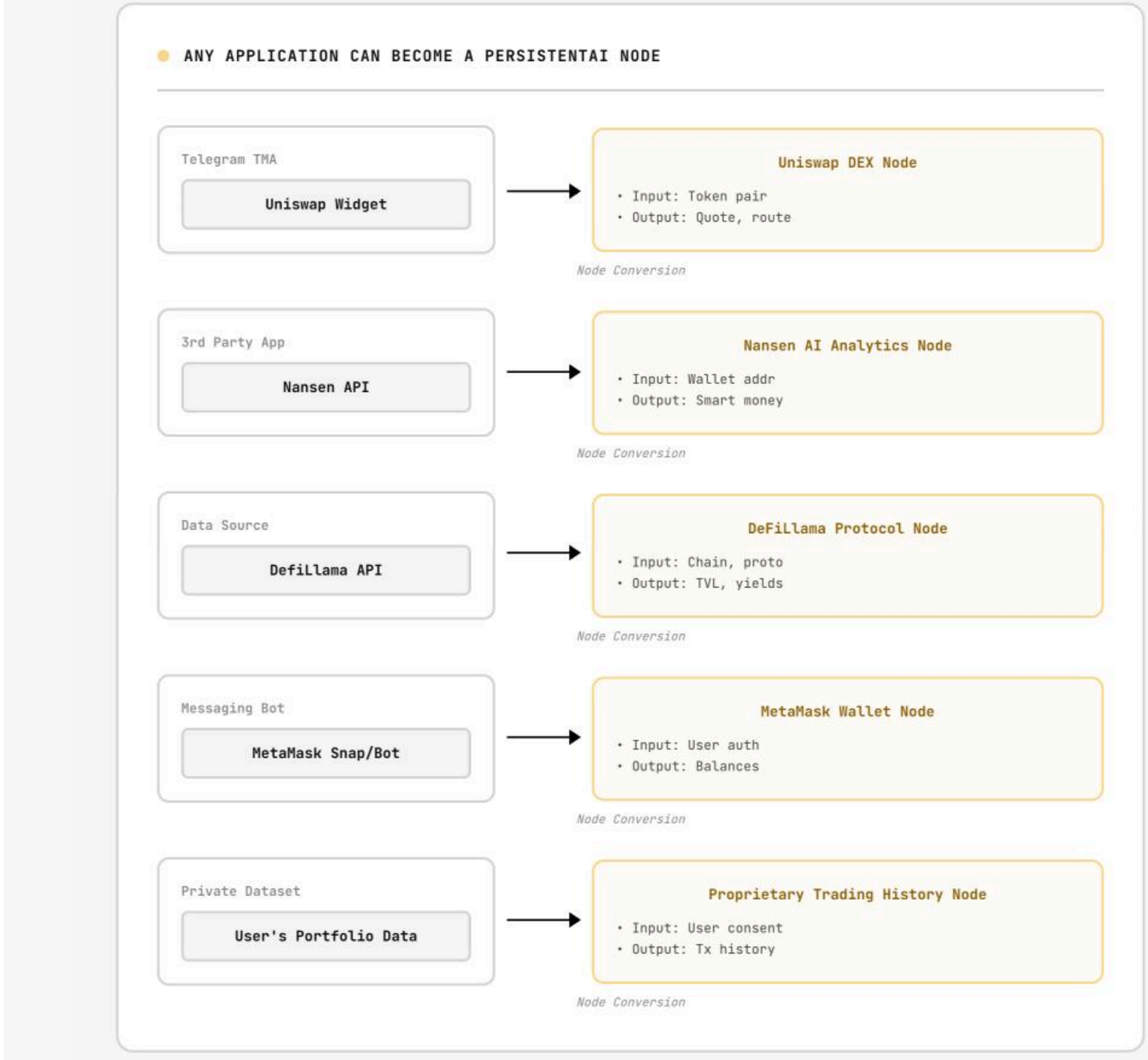

Fig4: Turning TMAs and 3rd Party Apps into Nodes

→ **This is UIO in practice.**

- **The AG-UI Protocol:** Rather than static interfaces, the AI Agents propose context-aware UI components-such as charts, forms, or product cards-that are rendered in real-time within the Telegram interface, creating a seamless and intuitive AX. Critically, every front-end element is a deterministic wrapping around the existing output nodes' ports and fields - not bug-prone agentic code.

→ **A simple takeaway is that when turned into PersistentAI nodes, TMAs functionality can be composed into a unified execution graph, augmented with 3-rd party data sources and capabilities. At the same time, the output nodes of the said graph can themselves be rendered to have generative interfaces.**

The AG-UI Protocol: Generative Interfaces

The **Agent-Generated UI (AG-UI) Protocol** represents a paradigm shift in how interfaces are constructed. Traditional applications have static interfaces defined at compile time. The developer decides what buttons exist, where they're located, and what happens when clicked. Users adapt to whatever interface the developer created.

AG-UI inverts this relationship. The AI Agent analyzes:

1. User intent expressed in natural language
2. Current context (conversation history, user preferences, environment)
3. Available data (from databases, APIs, blockchain state)
4. Appropriate visualization methods

Based on this analysis, the Agent proposes UI components dynamically. These proposals take the form of structured JSON describing the component type, data to display, and interactive behaviors. The frontend receives these proposals, validates them against allowed component types (for security), and renders them in real-time.

This creates several advantages:

1. **Context Awareness:** The same query produces different interfaces based on context. "Show me my portfolio" might render a summary card for a quick check or a detailed dashboard when the user is in analysis mode.
2. **Data-Driven Design:** UI components reflect actual data structure rather than forcing data into predetermined templates. If a user's portfolio has three tokens, the interface shows three items. If they have fifty tokens, the interface adapts (perhaps with grouping, filtering, or pagination).
3. **Progressive Disclosure:** The Agent can start with a simple interface and add complexity only when needed. This prevents overwhelming users with options while keeping advanced functionality accessible.
4. **A/B Testing at Scale:** Different users can receive different UI proposals, and the system can learn which interfaces lead to better outcomes (task completion, user satisfaction, conversion).
5. **Multilingual by Default:** Since the UI is generated from structured data rather than hardcoded text, localization becomes trivial. The same backend can generate interfaces in any language.

- **UIO for Paradigm Shift in User Experience:** integrating data and TMAs into PersistentAI allows for a single user intent to trigger a specific execution across the necessary nodes. Every interaction and execution step is registered on TetraChain, ensuring that the cost, source, and, ultimately, intent of each inference is transparent, auditable, and can be formally verified and monetized.

- **Guaranteed Durability via DBOS:** Robustness is guaranteed through Durable Execution (via DBOS), which ensures that complex Agentic flows survive infrastructure failures and maintain "exactly-once" operation semantics. This is critical for mission critical environments, such as financial operations (banking apps), operations with irreversible blockchain transactions, composing structured products and other similar applications.

Durable Execution via DBOS

The technical foundation enabling reliable UIO is **DBOS (Database-Oriented Operating System)**, a durable execution framework that guarantees exactly-once operation semantics. This is crucial for high-stakes operations like financial transactions.

Traditional server applications are ephemeral. If the server crashes mid-transaction, the state is lost. The system might:

1. Complete part of a multi-step operation, leaving the user in an inconsistent state
2. Retry the entire operation, potentially charging the user twice
3. Fail silently, requiring manual intervention to determine what happened

DBOS eliminates these failure modes through continuous checkpointing. Every step of every workflow is persisted to PostgreSQL before proceeding to the next step. This creates several powerful guarantees:

1. **Exactly-Once Execution:** Even if the server crashes and restarts multiple times, each operation executes exactly once. Idempotency is guaranteed at the system level rather than requiring developers to implement it in application code.
2. **Durable Waiting:** An Agent can wait indefinitely for external events-user confirmations, blockchain transactions, webhook callbacks-without occupying server resources. The workflow state is persisted, and execution resumes automatically when the awaited event occurs.
3. **Time-Travel Debugging:** Because every state transition is recorded, developers can "replay" any execution to understand exactly what happened. This is invaluable for debugging complex multi-step operations.
4. **Automatic Recovery:** Infrastructure failures (server crashes, network partitions, database restarts) are handled transparently. Workflows resume exactly where they left off with no data loss.
5. **Verifiable Execution:** The complete execution trace can be exported and verified by third parties. This is essential for compliance, auditing, and dispute resolution.

- **X402, RAG, AP2:** Agentic Each node or AI Agent is equipped with an x402 module, facilitating an "Agent Economy" through instant payments. For example, a creator who develops a specialized Twitter RAG (Retrieval-Augmented Generation) Agent can monetize it directly by receiving, for example, 10% of every inference cost that utilizes the data one has stored and offered for use. Every time a user triggers an inference from that Agent, the creator receives funds instantly. PersistentAI acts as the "Facilitator" or "Resource Server" in the x402 flow.

- **The ultimate "end product" is one unified chat interface where the user simply expresses their intent.** Instead of navigating a fragmented landscape of individual bots - the old reactive paradigm, the user interacts with a single, intelligent thread powered by the collective intelligence of the entire Telegram ecosystem - the new soon to be dominant **proactive (or intent-driven) paradigm.**

PERSISTENTAI ORCHESTRATION FLOW

(Intent-Driven Generative UI via MCP Standard)

STEP 1: Multi-Source MCP Servers (TMAs, Analytics, DeFi)

Blum [TMA]

Nansen [Analytics]

TONScan [Explorer]

Wallet [TMA]

DeDust [DEX]

FireFlow: MCP ADAPTER
(Universal Tool & Resource Normalization)

Atomic Action Node Library (12 Nodes)

get_balance	get_portfolio_composition	scan_yield_opportunities	get_token_prices
calculate_risk_score	estimate_gas	simulate_swap	claim_rewards
swap_tokens	provide_liquidity	transfer	get_analytics_insights

STEP 3: Output Nodes → Generative UI

User: "Optimize my yield across chains with <5% risk"

Intent Parser (LLM)

Wallet Balance [Blum]

Protocol APY Scan [DeDust] [STON.fi]

Gas Estimator [TON]

Risk Scorer [Risk] [Calc]

Strategy Optimizer

- Route Planning
- Risk Assessment
- APY Calculation

Output Nodes

(Data)

LLM Reasoning

(Process Data)

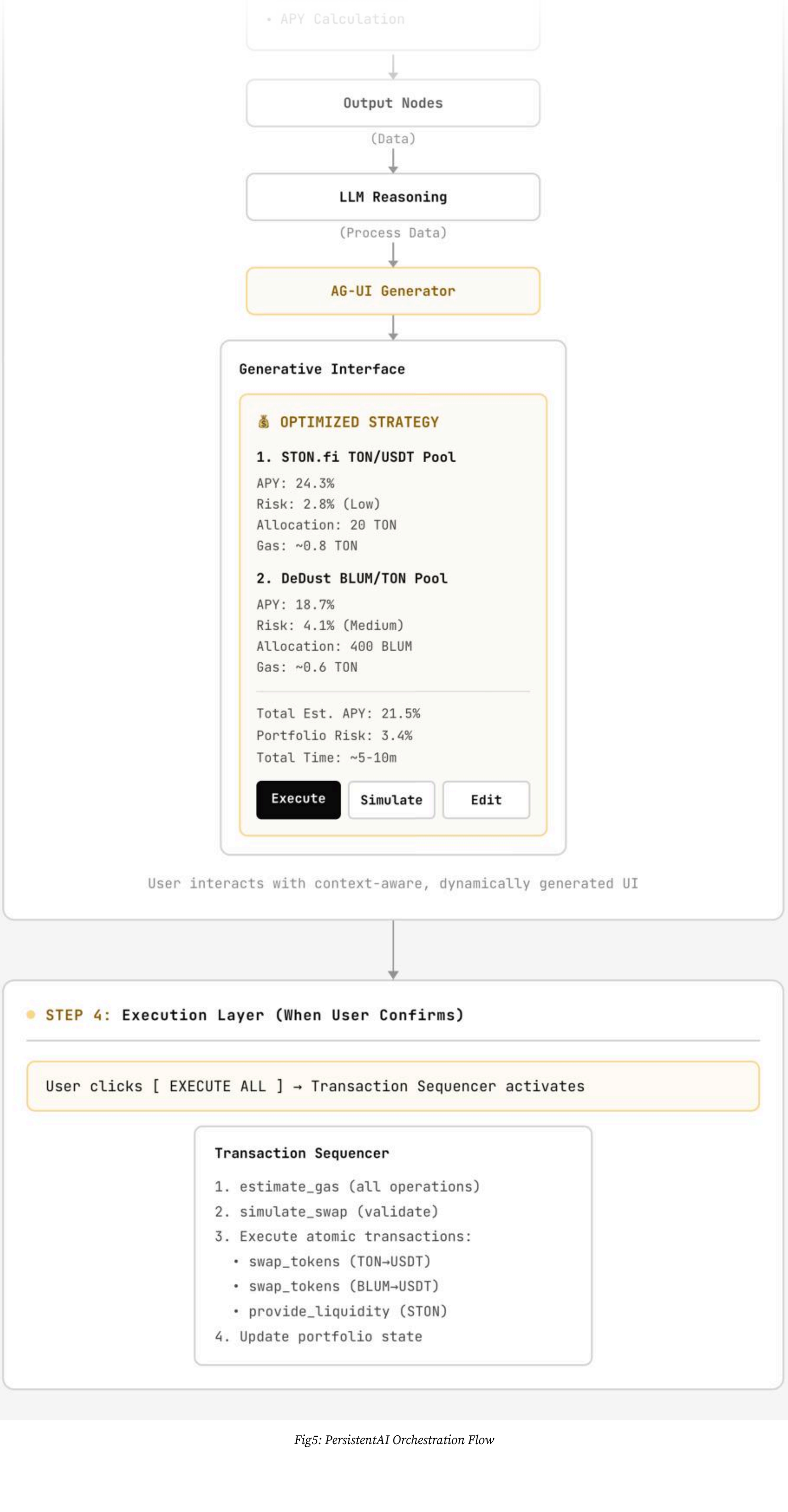

Fig5: PersistentAI Orchestration Flow

7. **PersistentAI Enables Inference Tokenization:** By enabling the tokenization of Agents, individual nodes or datasets as NFTs, PersistentAI enables a marketplace for data and intelligence. Technically, every inference graph is decomposed into individual nodes, their inputs (like RAG'ed twitter feed for example) and outputs. Every node in the execution graph has a x402 port allowing it to act as an independent economic agent - every time an inference is made through it, a payment distribution is triggered across the creator, the current owner and other participants in the value chain, such as data providers. Transactions are almost instantaneous thanks to high throughput TetraChain.

   This allows for price discovery, royalties and other revenue-sharing models rewarding creators efficiently, fairly and instantly.

   The fee is automatically split via the x402 payment protocol across all contributors. An example split could be:

   - **50%** to to model providers (Anthropic, OpenAI, Cocoon etc. for LLM API calls)
   - **10%** to node creators (developers who built the individual nodes used in the Agent's flow)
   - **10%** to node owners (those owning the node NFT)
   - **15%** to original agent creator
   - **10%** to data providers (APIs, oracles, blockchain indexers that supply data)
   - **5%** to protocol treasury (ecosystem development and grants)

The Agent Economy: Inference Tokenization

The Participation Economy

The most successful platforms of the coming decade will not merely provide superior user experiences-they will enable users to participate in value creation and capture. This is the core thesis of the **participation economy:** users should not only consume services, but also benefit economically from their engagement and contributions.

PersistentAI embeds this principle at the architectural level. Every interaction generates value that flows to multiple stakeholders:

1. **Users** receive intelligent services that save time and improve decisions
2. **Creators** earn revenue when their Agents/ Nodes are used
3. **Node operators** receive compensation for providing computational infrastructure
4. **The protocol** sustains itself through a modest treasury allocation

This creates aligned incentives across the ecosystem. Creators are motivated to build high-quality Agents because they profit from usage. Users benefit from a competitive marketplace of Agents. Node operators ensure infrastructure reliability because their revenue depends on uptime. The result is a self-sustaining economic system that grows more valuable as adoption increases.

The Problem with Traditional Monetization

As discussed earlier, traditional monetization models for TMAs are inadequate. But the problem runs deeper than mere technical limitations-it's a fundamental misalignment of incentives.

When monetization requires degrading user experience (ads, paywalls, data harvesting), creators face an impossible choice: build something users love and go broke, or implement monetization tactics that alienate your audience. This tension has plagued the internet for decades, leading to:

1. The "attention economy" where apps compete to be addictive rather than useful
2. Privacy violations as user data becomes the product
3. "Dark patterns" that trick users into unwanted subscriptions

PersistentAI's Agent NFT economy solves this by aligning all stakeholders through transparent, usage-based revenue sharing.

Agent NFTs: Liquid Ownership of Intelligence

When a creator builds an AI Agent on PersistentAI and chooses to monetize it, the Agent is minted as an **NFT (Non-Fungible Token)** on the TetraChain blockchain (a Layer 2 on TON). This NFT represents ownership rights to the Agent's future revenue stream.

The economic structure is straightforward and transparent:

When a user interacts with an Agent and triggers an inference (e.g., asking for product recommendations, requesting market analysis, executing a trade), a small fee is charged. This fee is automatically split for example as follows:

1. **50%** to to model providers (Anthropic, OpenAI, Cocoon etc. for LLM API calls)
2. **10%** to node creators (developers who built the individual nodes used in the Agent's flow)
3. **10%** to node owners (those owning the node NFT)
4. **15%** to original agent creator
5. **10%** to data providers (APIs, oracles, blockchain indexers that supply data)
6. **5%** to protocol treasury (ecosystem development and grants)

This structure creates several powerful dynamics:

Sustainable Creator Income: Creators earn revenue proportional to how useful their Agents are. A highly valuable Agent generates recurring income without requiring the creator to engage in salesmanship, advertising, or aggressive monetization tactics.

Tradeable Assets: Agent NFTs can be bought and sold on open marketplaces. If a creator builds an Agent that becomes popular but wants to cash out, they can sell the NFT to another party. The purchasing party evaluates the Agent's usage metrics and projects future revenue to determine a fair price.

Incentive Alignment: Because the original creator retains a royalty, they benefit even after selling the NFT. This means they're incentivized to keep the Agent's reputation intact and potentially contribute to its ongoing development.

Transparent Valuation: All usage metrics are on-chain and publicly auditable. Anyone evaluating an Agent NFT can see:

1. Total number of inferences executed
2. Revenue generated over various time periods
3. User retention metrics (how many users return to the Agent)
4. User satisfaction scores (if reputation systems are implemented)

Composability: Since Agents can call other Agents, revenue flows through the system. If Agent A uses Agent B as a component, Agent B's NFT holder earns a share of the transaction. This incentivizes building high-quality, reusable components.

The x402 Payment Protocol

To enable instant micropayments for Agent inferences, PersistentAI integrates the **x402 payment protocol** (originally developed by Coinbase). This protocol facilitates instant, low-friction payments between Agents and users, as well as between Agents themselves.

Traditional payment systems introduce latency (credit card processing takes days to settle) and high fees (merchant fees of 2-3% make micropayments uneconomical). Cryptocurrency transactions are faster but still introduce friction-users must sign transactions, pay gas fees, and wait for block confirmation.

The x402 protocol solves this through payment channels and optimistic settlement:

1. Users deposit funds into a payment channel when first interacting with an Agent ecosystem
2. Individual inference fees are deducted instantly from this channel balance
3. Settlement to the blockchain happens periodically in batches, minimizing transaction costs
4. Dispute mechanisms ensure users can't be cheated even with optimistic settlement

For users, this means:

1. No transaction signing for every inference
2. Predictable costs disclosed upfront
3. Ability to set spending limits
4. Instant access to Agent capabilities

For creators, this means:

1. Instant revenue (not waiting for monthly payouts)
2. Micropayments that would be uneconomical with traditional systems become viable
3. Automatic split payment to all stakeholders
4. No payment processor fees eating into revenue

Network Effects and Market Dynamics

The Agent NFT economy creates powerful network effects that accelerate ecosystem growth:

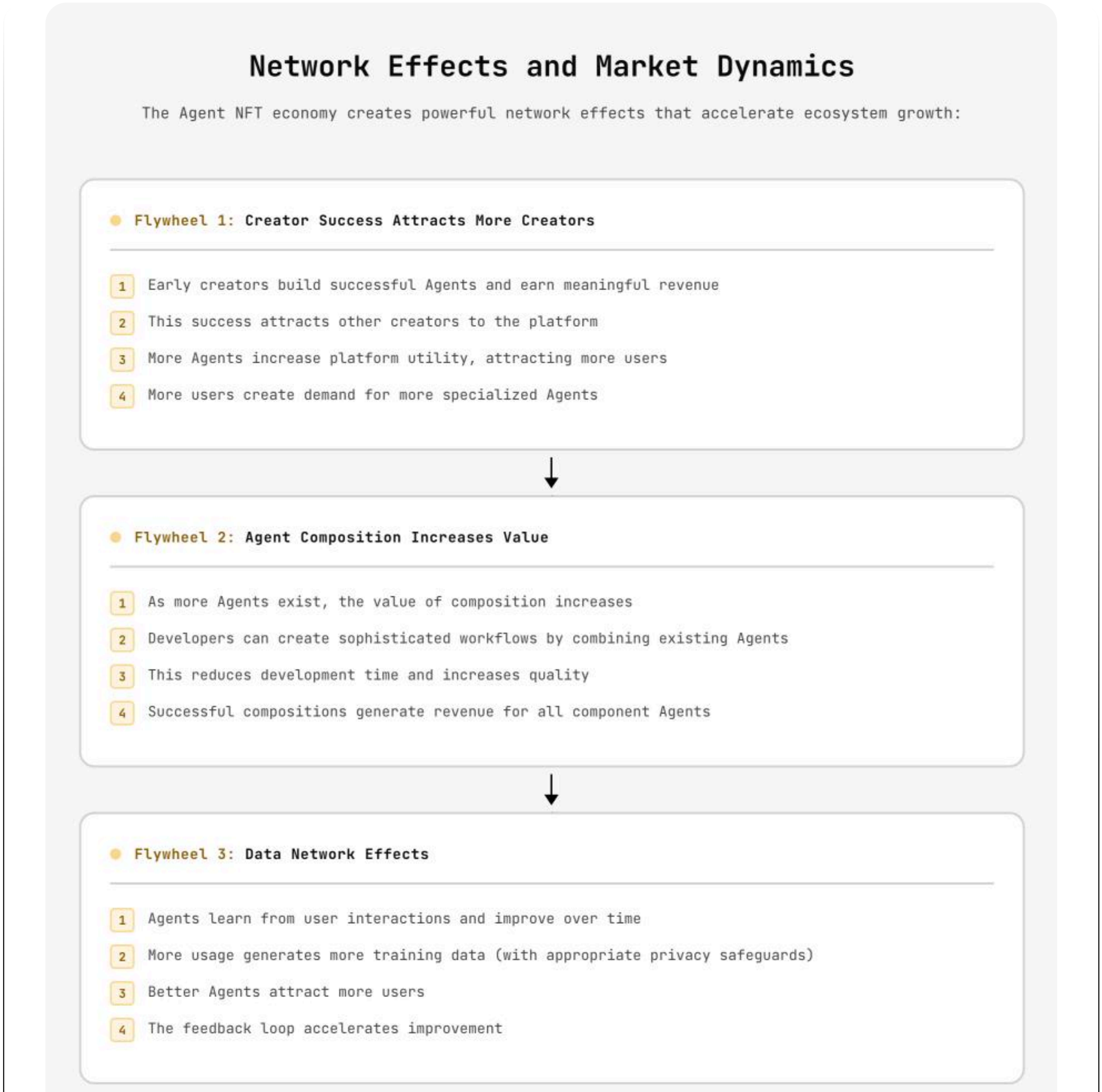

Fig6: Network Effects and Market Dynamics

Market Maturation: Over time, we expect several market structures to emerge:

1. **Agent Marketplaces:** Platforms for discovering, evaluating, and purchasing Agent NFTs
2. **Agent Portfolios:** Entities that own diversified collections of Agent NFTs, similar to intellectual property portfolios
3. **Agent Improvement Services:** Specialized developers who purchase underperforming Agents, improve them, and capture value through the equity upside
4. **Agent Derivatives:** Financial instruments based on expected Agent revenue (similar to music royalty securitization)

REVENUE DISTRIBUTION PER INFERENCE

(Smart Money Tracking Agent)

USER TRIGGERS INFERENCE

"Alert me when top Ethereum wallets accumulate tokens similar to my portfolio"

↓

Fee: $2.50

x402 Payment Module
(Instant Distribution)

↓

Agent Owner	Node Creators	Model Provider	Data Providers	Protocol Treasury
$1.00	**$0.75**	**$0.375**	**$0.25**	**$0.125**
(40%)	(30%)	(15%)	(10%)	(5%)
Built the smart money tracker	Created specialized nodes	Claude, GPT-4, etc	Nansen, Dune Analytics	Ecosystem development

EXAMPLE: Smart Money Tracker Agent Flow

NODES USED		REVENUE SPLIT
1. LLM Node (Claude)	→	**15% to Anthropic**
2. Nansen Wallet Tracker	→	**5% to Nansen**
3. Dune Analytics Query	→	**5% to Dune**
4. Portfolio Similarity Calc	→	**15% to Node Creator A**
5. Pattern Recognition ML	→	**15% to Node Creator B**
6. Alert System	→	**0% (platform node)**

Agent Owner (orchestrated flow)	**40% of inference fee**
Protocol Treasury	**5% of inference fee**

COMPOSABILITY EXAMPLE:

This Smart Money Agent can become a node itself:

Another developer builds "Auto-Copy Trading Bot"
→ Uses Smart Money Agent as a component
→ Original Agent owner earns from every copy trade
→ Creates recursive value flow

Fig7: Revenue Distribution per Inference

8. The Unified Intent Interface: Orchestrating Agent and Secure Fulfillment

The architecture of PersistentAI consolidates fragmented bot ecosystems into a **Unified Chat Interface**, replacing the friction of switching between disparate applications with a continuous, context-aware thread. An **Orchestration Agent** acts as the central "General Manager" to streamline this process:

- **Intent Decoding:** It parses multi-layered user requests to understand complex financial or logical goals beyond simple keyword recognition.
- **Modular Delegation:** It programmatically coordinates specialized "Worker Agents"-such as data indexers or settlement rails-to execute a sequence of actions.
- **Dynamic UX:** Using the AG-UI protocol, it renders real-time interface components, like transaction cards or charts, directly within the chat as needed.

The system achieves "smart" fulfillment through **Trusted Execution Environments (TEE)**, which enable high-fidelity intelligence while maintaining strict privacy. This secure layer ensures that user intent is handled with both precision and transparency:

- **Private Data Integration:** The agent analyzes sensitive records, such as personal financial history, without exposing raw data to LLM providers or external networks.
- **Proactive Intelligence:** By combining private context with real-time blockchain state data, the system can suggest or execute optimized actions based on a holistic understanding of the user's situation.
- **Formal Verification:** Every step is recorded on TetraChain, providing a formally verified and auditable trail that ensures actions remain strictly aligned with the original user intent.

Conclusion: The Path Forward

1. PersistentAI addresses a fundamental deficiency in the contemporary AI stack: the absence of a deterministic orchestration layer between probabilistic foundation models and end-user applications. Without it, AI agents remain unreliable for any operation involving money, identity, or irreversible state changes. PersistentAI provides that layer - and in doing so, introduces what we term User Intent Orchestration (UIO): the paradigm shift from reactive navigation across fragmented applications to proactive, intent-driven fulfillment via autonomous economic agents.

2. The architecture is purpose-built for deterministic execution in high-stakes contexts. The Haskell core enforces formal verification at the type level. DBOS guarantees exactly-once execution semantics, eliminating the class of failures - partial transactions, duplicate charges, orphaned states - that make current agent frameworks unusable in financial contexts. Trusted Execution Environments (TEE) enable private inference on sensitive portfolio and identity data without exposure to LLM providers or third-party tool operators. These are not features added to a chatbot; they are foundational properties of an execution engine designed for economic activity.

3. PersistentAI resolves Telegram's ecosystem fragmentation through a bidirectional transformation: any TMA, data source, or external service becomes a composable Tool or MCP with defined I/O ports and an x402 payment endpoint - while any agentic flow built on the platform can itself be deployed as a TMA. Tools and MCPs compose into unified execution graphs, wherein a single user intent triggers deterministic traversal across all requisite capabilities. The AG-UI protocol renders context-aware interfaces directly in Telegram chat as deterministic wrappers around tool outputs - not probabilistically generated code. The result is that a user types a complex, multi-constraint request into a single Telegram thread, and the orchestration engine decomposes it into parallel execution across dozens of autonomous tools, settles micropayments to every economic participant via x402, and renders an interactive, auditable result - all without the user leaving the chat.

4. Each new Tool or MCP added to the catalog increases what the orchestration engine can compose. Each new Agent deployed as a TMA generates x402 revenue for its component creators, incentivizing further specialization. Every tool in every execution graph functions as an independent economic agent - charging per-inference fees that are automatically split across model providers, tool creators, Agent NFT holders, data providers, and the protocol treasury. All settlement flows through TetraChain (TON L2), which provides sub-second finality, immutable audit trails for every execution step, and the throughput required for agentic micropayments at scale.

5. The immediate product surface is Telegram-native DeFi and commerce - cross-chain portfolio rebalancing, TON-native agentic commerce via TMAs, intent-driven yield optimization, and portfolio risk management. These are not hypothetical scenarios; they represent the concrete proving ground where the orchestration engine, settlement infrastructure, and generative UI converge into experiences that no fragmented collection of interfaces/bots can replicate.

6. The old paradigm of navigating fifty tabs, ten apps, and three wallets collapses into a single continuous conversation. With PersistentAI, Telegram transcends its chat origins to become the Universal Intent Interface.

Appendix 1: Use Cases in Detail

Use Case 1

Cross-Platform DeFi Portfolio Rebalancing with Privacy-Preserving Execution

A retail crypto investor holds assets across three chains (TON, Ethereum, Base) and uses multiple Telegram bots to track prices, monitor yield, and execute swaps. Today, rebalancing requires manually checking each position, comparing yields across protocols, calculating optimal allocation, and executing 6-8 separate transactions - a process that takes 30-60 minutes and is prone to slippage and human error.

With PersistentAI, the user states a single intent: *"Rebalance my portfolio to 40% stables, 35% blue-chips, 25% yield-bearing positions - minimize fees."*

The Orchestration Agent decomposes this into a coordinated execution graph:

1. **Portfolio Aggregation Tools** pulls balances from connected wallets across all three chains via existing MCP Integrations (e.g., Blum, Nansen, @Wallet).

2. **Price Discovery Tools** queries DEX aggregators and oracles for real-time pricing and liquidity depth.

3. **Optimization MCP** calculates the minimum number of swaps required to achieve target allocation, accounting for gas costs, bridge fees, and slippage tolerance.

4. **TEE-Protected Execution** processes the user's full portfolio context - including historical positions and risk preferences - within a Trusted Execution Environment, ensuring that no LLM provider or third-party node sees the complete financial picture.

5. **Settlement Tools** help to batch approved transactions through the ASL, splitting execution across chains and settling via x402 micropayments to each node operator in the flow.

6. **AG-UI Rendering** displays a before/after allocation chart, a fee breakdown card, and a single confirmation button - all generated dynamically within the Telegram chat interface.

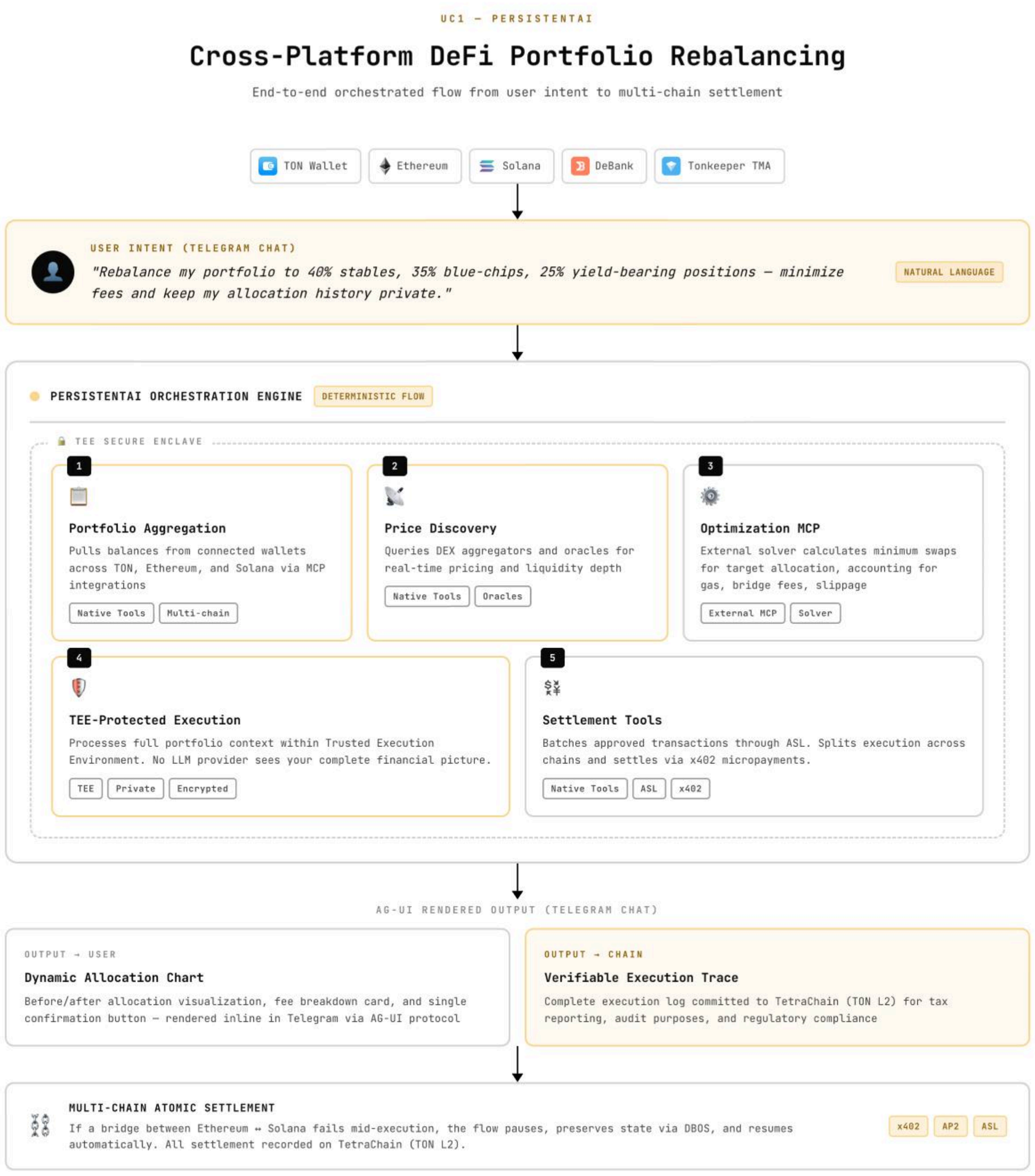

Fig8: Use-case 1

Every step is checkpointed via DBOS. If the bridge between Ethereum and TON fails mid-execution, the flow pauses, preserves state, and resumes automatically when connectivity is restored - the user's funds are never left in an inconsistent state. The complete execution trace is committed to TetraChain, creating a verifiable record for tax reporting or audit purposes.

Revenue distribution per inference: The Portfolio Aggregation MCP creator earns their x402 share, the Optimization Node creator earns theirs, and the user pays a single bundled fee that is transparent and disclosed upfront. If any of these nodes are tokenized as Agent NFTs, secondary owners earn proportional revenue from every rebalancing event.

Use Case 2

A Telegram-native user runs a bot-based business and needs to purchase infrastructure - specifically, 3 months of premium hosting from a TON-native provider. Today, this means manually browsing multiple TMA storefronts, comparing prices denominated in different Jettons, checking whether providers accept USDT on TON or require a swap, copying wallet addresses, initiating a transfer via Tonkeeper, and then screenshotting the transaction hash as a "receipt." There's no unified search, no automated price comparison, no compliance record, and no single thread tying intent to fulfillment.

With PersistentAI, the user states a single intent: *"I need 3 months of premium hosting for my Telegram bot. Find the cheapest TON-native provider and pay with USDT on TON. Get me a receipt."*

The Orchestration Agent decomposes this into a coordinated execution graph across the TON ecosystem:

1. **TMA Discovery MCP** queries TON-native service provider storefronts via MCP. It resolves merchant TON DNS names, checks on-chain reputation through Soulbound Tokens (SBTs), and pulls real-time pricing directly from storefront smart contracts. Because PersistentAI operates natively within Telegram, this MCP-connected tool can access TMA catalogs that are invisible to traditional web crawlers - these are Telegram-native storefronts that only exist within the Mini App ecosystem.

2. **Price Comparison Tools** normalize quotes across Jetton trading pairs (USDT/TON via STON.fi, DeDust). They factor in current swap slippage, TON gas costs, and merchant discount tiers for bulk or prepaid purchases. If the cheapest provider prices in TON but the user holds USDT, the tools calculate the all-in cost including the swap, so the comparison is apples-to-apples.

3. **TON Connect Auth + Compliance MCP** authenticates the user via TON Connect wallet signature. If the selected merchant requires identity verification (increasingly common for infrastructure providers in regulated jurisdictions), this external MCP generates a ZK proof from the user's Galactica zkCert - proving eligibility (e.g., jurisdiction, not on a sanctions list) without exposing any personal data to the merchant. The merchant receives a cryptographic attestation of compliance, not the user's documents.

4. **AG-UI Rendering** presents the top provider quotes as interactive cards directly in the Telegram chat interface. Because the system has detected this is a service purchase (not a physical good), it displays uptime SLAs, bandwidth specs, and support tier alongside price. The Jetton balance is shown inline, with a swap preview if needed, and a one-tap "Pay with TON USDT" confirmation button. The entire UI is rendered contextually via the AG-UI protocol - no redirect, no browser, no external app.

5. **Jetton Payment Tools** handle the x402 payment authorization flow. The user signs a spending authorization with their TON wallet - a message that grants the selected merchant permission to withdraw a specific token amount within a defined timeframe. The merchant then executes the transfer against this authorization. If the user holds the wrong Jetton denomination, the tools auto-swap through STON.fi as a prerequisite step before the authorization is signed. The settlement is atomic: either the merchant successfully executes the authorized transfer and the service activation is confirmed, or the authorization expires unused. No partial states, no orphaned payments.

6. **TetraChain Record Tools** record the full transaction bundle on TetraChain (TON L2): the payment proof, the service activation receipt, the merchant's SBT verification, and the compliance attestation. This creates an immutable, verifiable, and queryable record of the entire transaction lifecycle. Both the user and the merchant can reference this record for disputes, accounting, or regulatory audits.

UC2 — PERSISTENTAI

TON-Native Agentic Commerce via Telegram Mini Apps

From natural language intent → TMA storefront discovery → Jetton payment → TetraChain (TON L2) settlement

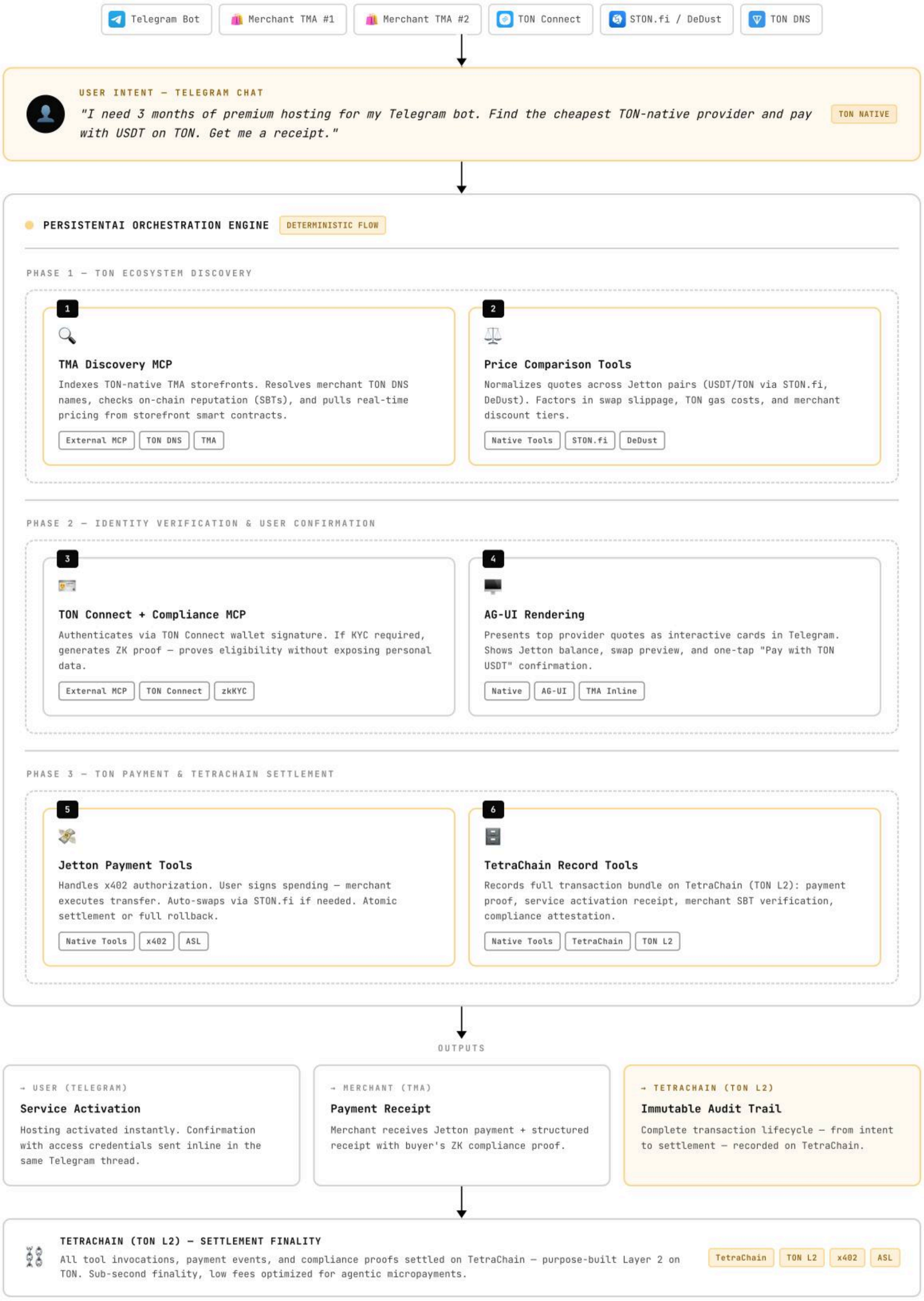

Fig9: Use-case 2

Every step is checkpointed via DBOS. If the Jetton transfer encounters a temporary network congestion spike on TON, the flow pauses at the payment step, preserves the full execution state, and retries automatically - the user is never charged twice, and the service is never activated without confirmed payment. The durable execution guarantee is critical here because Telegram users expect instant, reliable transactions within the chat UX. Any failure that requires the user to "try again" or manually verify destroys the agentic experience.

The entire flow executes within a single Telegram thread. The user never leaves the chat, never manually compares TMA storefronts, never copies wallet addresses, and never chases a receipt. The hosting is activated, the credentials are delivered inline, and the verifiable proof of everything - from intent to settlement - lives permanently on TetraChain.

Use Case 3

Seeking Yield in the TON Ecosystem

A Telegram user holds a mixed portfolio - 55.4 TON, 1,200 BLUM, and 3.2K USDT - across their connected wallet. They're earning a passive 8.2% APY and suspect they're leaving yield on the table, but have no idea which pools across STON.fi, DeDust, or TON Validators would give them a better return for their specific holdings. Today, figuring this out means manually checking each DEX, comparing APYs that change hourly, mentally calculating how much of each token to allocate, estimating gas costs, and assessing whether the pool TVL is large enough to trust.

With PersistentAI, the user types: "*I want to optimize my yield. Show me the best opportunities and rebalance.*"

The Intent Parser (LLM) classifies this as a Yield Optimization intent and decomposes it into a parallel execution graph - this is a critical architectural point: different intents produce different execution graphs, which produce different UIs. The same engine that handles risk management produces an entirely different node composition and rendered output here.

Branch A (Portfolio Analysis) executes get_balance and get_portfolio_composition atomic action nodes simultaneously with **Branch B (Market Scanning)**, which executes scan_yield_opportunities and get_token_prices. Branch A pulls current holdings with USD valuations and allocation percentages. Branch B queries STON.fi, DeDust, and TON Validators for live APYs, TVLs, pool compositions, and minimum stake requirements. Both branches run concurrently - the parallel execution is not cosmetic, it halves the data aggregation latency.

The merged outputs feed into the **Risk Assessment Engine** (calculate_risk_score), which evaluates concentration risk, token volatility, pool TVL stability, and smart contract audit status for each candidate opportunity. This risk context then flows into the **Strategy Builder (LLM Reasoning)**, which synthesizes portfolio state + yield opportunities + risk scores into a ranked set of actionable recommendations. It calculates optimal allocations, projected APY deltas, and the specific swap paths required.

The Generative UI Renderer then produces a **custom interactive dashboard directly inside the Telegram chat** via the AG-UI protocol. This isn't a static message - it's a dynamically generated interface showing the user's current portfolio composition, risk score, and three ranked yield opportunities (STON.fi TON/USDT at 24.3%, DeDust BLUM/TON at 18.7%, TON Validators Staking at 5.2% - greyed out because the user doesn't meet the 10K TON minimum). Each opportunity has SIMULATE and PROVIDE LP action buttons. A 30-day projection bar shows the delta: current strategy yields +$25, the optimized strategy yields +$58 - a 2.3x improvement.

When the user clicks EXECUTE ALL, the **Transaction Sequencer** activates: estimate_gas for all operations → simulate_swap to validate each trade before committing → atomic execution of all swaps and LP provisions. The full execution trace is recorded on TetraChain (TON L2). DBOS checkpointing ensures no partial states - if one of the LP provisions fails after a swap succeeds, the entire batch rolls back.

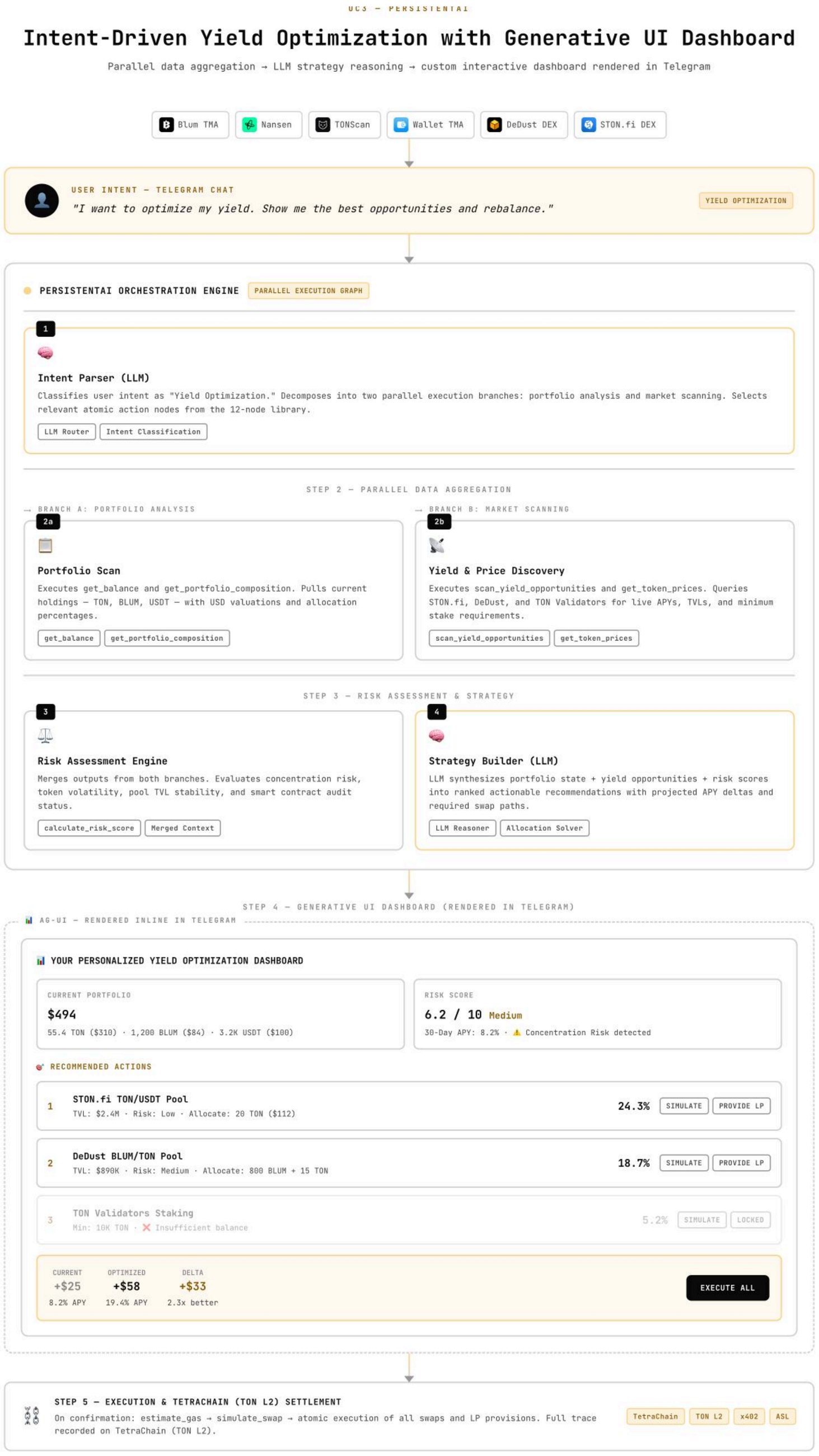

Fig10: Use-case 3

Appendix 2: UIO Explained

From User Experience to UIO - the Agentic Experience

Traditional user experience (UX) design focuses on optimizing static interfaces-buttons, forms, navigation hierarchies. This paradigm worked well for the desktop and early mobile eras when applications were self-contained and interactions were predictable. However, as AI capabilities advance, we're witnessing a fundamental shift in how humans interact with digital systems.

User Intent Orchestration (UIO) represents a new paradigm where users express intent rather than navigate interfaces. Instead of clicking through menus to find a product, compare prices, and complete checkout, a user simply states: "Find me running shoes under $100 with good arch support." The AI Agent interprets this intent, orchestrates the necessary operations, and presents results in a contextually appropriate format.

This shift is not merely aesthetic-it represents a fundamental change in the relationship between humans and software. In the UIO paradigm:

1. **Intent Replaces Navigation:** Users describe what they want rather than how to get it
2. **Context Drives Presentation:** The interface adapts to the user's situation, history, and preferences
3. **The Medium is Massively Integrated:** the entire web of data and apps must be integrated to arrive at truly holistic UIO
4. **Intelligence is Embedded:** The system understands nuance, handles ambiguity, and learns from interaction
5. **Actions are Autonomous:** Agents can complete complex multi-step operations without constant user supervision
6. **Context is Persistent:** The system constantly learns from every interaction. Private learning is, of course, preferred.

Why Telegram is the Ideal Platform for UIO

Telegram has evolved beyond its origins as a secure messaging platform to become what industry observers call an "Everything App." With over 500 million monthly active users, many of whom hold TON wallets, Telegram represents the largest ready-made distribution network for AI-powered applications. Users already conduct a wide range of activities within Telegram:

1. **Communication:** Personal and group messaging, channels, broadcasts
2. **Commerce:** Purchases through Telegram Mini Apps, peer-to-peer transactions
3. **Information:** News consumption, content discovery, community engagement
4. **Finance:** Cryptocurrency trading, DeFi interactions, payments

This makes Telegram uniquely positioned for the UIO revolution. Unlike standalone apps that require downloads, onboarding, and habit formation, TMAs exist within an environment users already inhabit daily. The barrier to adoption is dramatically lower-a user can interact with a sophisticated AI Agent simply by clicking a link shared in a chat.

Understanding the Current TMA Landscape

While Telegram has successfully cultivated a thriving ecosystem of Mini Apps, the user experience remains fundamentally fragmented. A typical user journey illustrates the problem:

Scenario: A user wants to research an investment opportunity and execute a token swap.

Current Reality:

1. Opens a market research TMA to analyze token fundamentals
2. Switches to a different TMA to check price charts and technical indicators
3. Navigates to yet another TMA to view community sentiment
4. Returns to the wallet TMA to check current holdings
5. Opens a DEX aggregator TMA to find the best swap rate
6. Finally executes the transaction through a separate interface

Each of these steps involves context switching-mentally tracking where you are, what you were doing, and what information you've already gathered. The user must manually integrate information across multiple disconnected interfaces. This cognitive overhead is the opposite of Agentic Experience.

The Developer's Burden

The fragmentation problem affects developers even more acutely than users. Building a TMA today requires:

Technical Complexity

1. Frontend development (React, Vue, or vanilla JavaScript)
2. Backend API development (Node.js, Python, Go, etc.)
3. Database design and management (PostgreSQL, MongoDB, Redis)
4. Authentication and authorization systems
5. Payment processing integration
6. Infrastructure deployment and scaling (Docker, Kubernetes, cloud providers)
7. Monitoring, logging, and error tracking
8. Security hardening and penetration testing

This technical stack creates significant barriers to entry. An enthusiast with a brilliant idea for a TMA might understand the business logic perfectly but lack the full-stack development skills to implement it. Even experienced developers face months of work to build production-ready infrastructure.

Operational Overhead

1. Server maintenance and monitoring
2. Security patches and updates
3. Scaling to handle usage spikes
4. Cost management across multiple cloud services
5. Debugging production issues
6. Managing database migrations and backups

These operational responsibilities never end. A successful TMA doesn't just need to be built-it needs to be maintained indefinitely.

The Monetization Vacuum

Perhaps most concerning is the absence of sustainable monetization models for TMA creators. The current options are limited and problematic:

1. **Advertising:** Clutters the interface, degrades user experience, and earns minimal revenue for small applications. Users have learned to ignore or actively block ads.
2. **Freemium Models:** Require building and maintaining two separate experiences (free and premium), managing subscription logic, and convincing users to upgrade. Conversion rates are typically below 5%.
3. **Transaction Fees:** Only viable for commerce or financial TMAs, and often require explicit fee disclosure that creates friction in the user experience.
4. **Sponsored Content:** Introduces conflicts of interest where the TMA creator's incentives may not align with user interests. Recommendations become suspect.
5. **Data Monetization:** Ethically questionable and increasingly regulated. Users are rightfully skeptical of applications that appear "free" but sell user data.

The result is a tragedy of the commons: talented developers invest significant time building TMAs but struggle to sustain themselves financially. This leads to:

1. Abandoned projects as creators move to paying work
2. Degraded quality as financial pressure mounts
3. Sketchy monetization tactics that alienate users
4. Consolidation toward larger players who can afford to operate at a loss

The ecosystem needs a monetization model that is transparent, non-intrusive, and directly tied to value creation. This is where PersistentAI's Agent NFT economy becomes transformative.

User Intent Orchestration: The Single-Thread Experience

The culmination of PersistentAI's architecture is User Intent Orchestration (UIO)-the ability for a user to express complex desires in a single thread and have multiple specialized Agents coordinate to fulfill that intent.

Consider a sophisticated scenario:

User: "I want to sell 20% of my crypto portfolio, use the proceeds to buy a stablecoin, and then find DeFi yield opportunities above 8% APY with minimal risk. Show me the best three options and execute the highest-yield one after I confirm."

In a traditional fragmented ecosystem, this would require:

1. Manual calculation of portfolio allocation
2. Checking current token prices across multiple exchanges
3. Executing multiple swaps (possibly on different platforms)
4. Researching DeFi protocols independently
5. Evaluating risk factors manually
6. Comparing APY after accounting for fees and slippage
7. Executing deposit into selected protocol

Each step has failure points, requires specialized knowledge, and introduces delays.

With PersistentAI's UIO, the user expresses intent once. Behind the scenes:

1. **Portfolio Analysis Subgraph** calculates current holdings and determines which tokens represent 20% of value
2. **Price Discovery Subgraph** queries multiple DEXes for current rates
3. **Optimization Subgraph** determines the best execution path to minimize slippage and fees (e.g. 1inch)
4. **DeFi Research Subgraph** (e.g. DefiLlama, Messari) scans yield protocols, filtering for 8%+ APY
5. **Risk Assessment Subgraph** evaluates smart contract security, historical performance, and liquidity depth
6. **Ranking Subgraph** scores options based on yield, risk, and user preferences
7. **UI Generation Subgraph** creates comparison cards for the top three options
8. **Execution Subgraph** awaits user confirmation, then orchestrates the multi-step transaction

Throughout this process, the user sees a single coherent interface updated in real-time. The complexity is hidden. Intelligence is embedded. The experience is seamless.

TetraChain (a Layer 2 blockchain on TON) powers the Agentic Settlement Layer (ASL): a high-throughput, low-latency payment rail for Agent-to-Agent, Human-to-Agent, and Agent-to-Human transactions, enabling trustless interaction with programmable money and DeFi. That is, every step is recorded on TetraChain, creating an immutable audit trail. If something goes wrong, the exact sequence of operations can be reconstructed. If there's a dispute, cryptographic proofs demonstrate what the Agent actually did.

Appendix 3: Technical Architecture Deep Dive

The Flow-Based Programming Paradigm

PersistentAI's development model centers on **visual flows** rather than traditional code. This represents a fundamental shift in how developers express computational logic.

In traditional backend development, developers write imperative code that explicitly defines every step. The developer must handle error conditions at each step, rollback logic if later steps fail, retry logic for transient failures, logging for debugging, metrics collection, authentication and authorization, and input validation. Even experienced developers frequently introduce bugs in this complexity.

With PersistentAI, the same logic is expressed as connected nodes in a visual interface. Each node is a self-contained unit with defined input and output types, built-in error handling, automatic logging, retry policies, and execution guarantees from DBOS.

Advantages of Visual Flows:

1. **Reduced Cognitive Load:** Humans process visual information faster than code. The flow structure makes the logic immediately apparent.

2. **Lower Barrier to Entry:** Non-programmers can understand and modify flows. Business analysts, designers, and domain experts can participate in development.

3. **Fewer Bugs:** Type-safe connections between nodes prevent many common errors. The flow won't execute if ports are connected incorrectly.

4. **Easier Testing:** Individual nodes can be tested in isolation. Entire flows can be tested by providing mock inputs.

5. **Visual Debugging:** When something goes wrong, developers can see exactly which node failed and inspect the data at that point.

6. **Versioning and Collaboration:** Flows are stored as structured data (JSON), making them easy to version control, diff, and merge using standard tools.

7. **Instant Deployment:** Changes to flows are deployed without traditional build-deploy cycles. Modify a flow, save it, and the new version is live.

The Node Catalog: 146+ Capabilities

PersistentAI provides a comprehensive catalog of pre-built nodes across 16 categories. Each node is a specialized capability that has been implemented, tested, and optimized by the platform team. This means developers don't need to implement common functionality from scratch.

Sample Node Categories:

1. **AI & Intelligence**

- LLM Call (multi-model support: GPT-4, Claude, Llama, etc.)
- Structured Output (force LLMs to return JSON with specific schemas)
- Chat History (maintain context across conversations)
- Tool Use (let LLMs decide which external tools to invoke)
- Vector Search (semantic similarity search across documents)
- Embeddings (convert text to vector representations)

2. **Blockchain & DeFi**

- Wallet Connect (TON wallet integration)
- Token Balance (query TON, Jetton, NFT balances)
- Token Transfer (send TON or Jettons)
- Jetton Swap (execute DEX trades)
- Contract Call (interact with arbitrary TON smart contracts)
- Event Listener (react to on-chain events)

3. **Data Transforms**

- JSON Parse/Build (convert between JSON and objects)
- Array Operations (map, filter, reduce, sort, group)
- Object Transform (reshape data structures)
- Type Conversion (string to number to boolean to date)
- Schema Validation (ensure data matches expected structure)

4. **Flow Control**

- Branch/Switch (conditional logic)
- Loop/Iterator (process collections)
- Parallel Execute (run multiple operations concurrently)
- Durable Wait (pause execution until an event occurs)
- Subflow Call (invoke other flows as subroutines)

This catalog is continuously expanding. The platform team adds nodes based on community requests. Advanced users can even create custom nodes and contribute them to the public catalog (earning node operator revenue when others use them).

Port-Based UI Binding

One of PersistentAI's most innovative features is treating UI components as first-class ports in the flow. This creates a reactive binding where interface elements update automatically when connected data changes.

Traditional architecture separates backend and frontend. The frontend must explicitly fetch data, parse responses, update local state, and trigger re-renders. This creates substantial boilerplate code and opportunities for desynchronization.

PersistentAI's port-based model connects flow node data output ports directly to UI component data input ports. When the node produces new data, it flows directly to the UI component. No manual state management. No fetch-parse-update cycle. The binding is declarative.

For example, a trading dashboard displays current token prices. The Price Feed node emits new prices as they arrive, and the chart updates automatically. If the chart component is removed from the UI, the connection is severed automatically. If multiple UI components need the same data, they all connect to the same output port-no additional fetching required.

This eliminates entire classes of bugs:

1. Stale data from caching issues
2. Race conditions from concurrent updates
3. Memory leaks from uncleared intervals
4. Desynchronization between multiple UI elements showing the same data

Comparison with Traditional Approaches

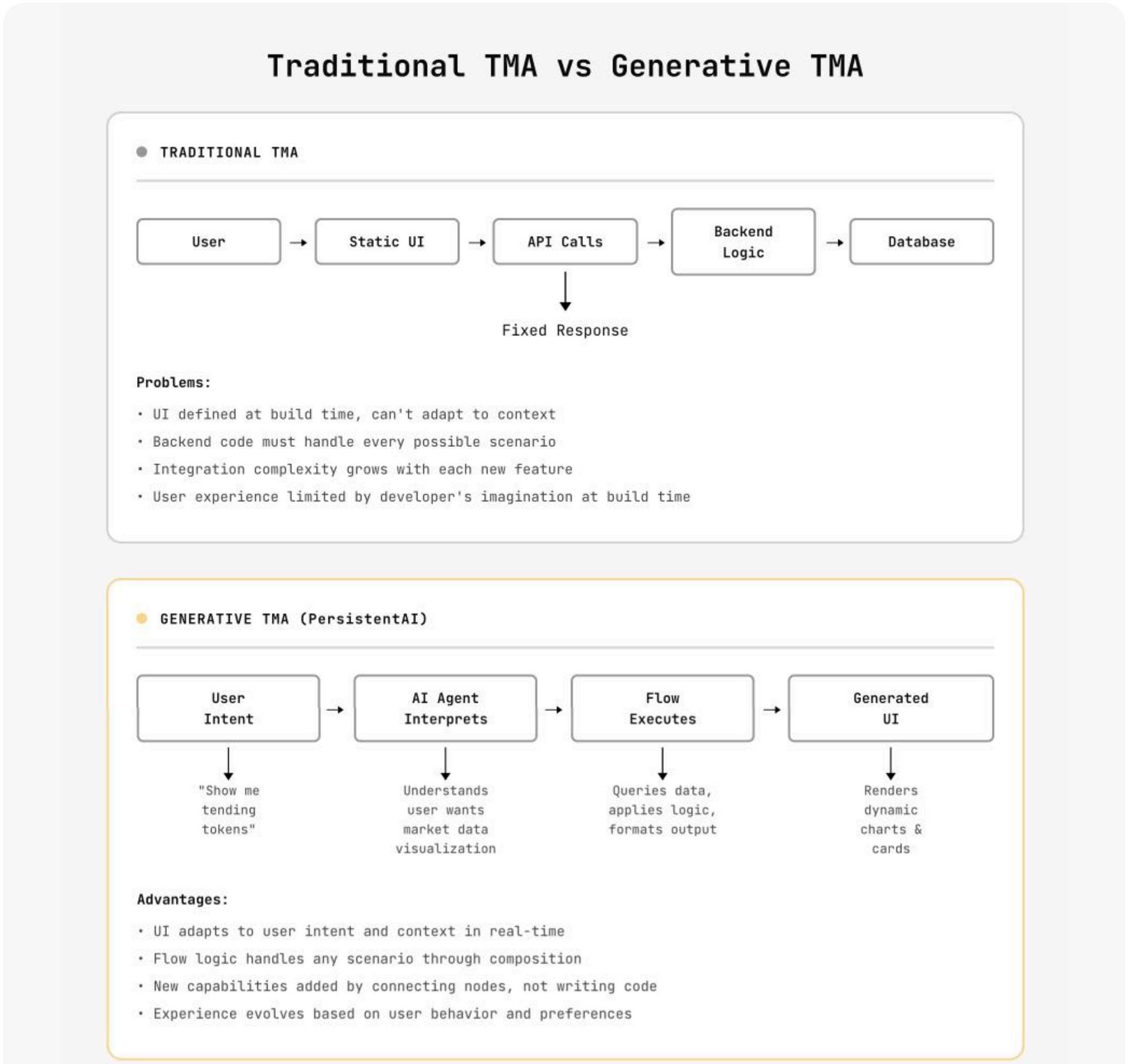

Fig11: Traditional TMA vs Generative TMA

The GenAI Platform Alternative

Some might argue: "Can't you just use LangChain, AutoGPT, or other Agent frameworks?"

These tools are valuable but solve different problems:

1. **LangChain/LlamaIndex:** These are libraries for building AI applications in code. They reduce boilerplate but still require developers to write Python or JavaScript code, manage execution infrastructure, handle persistence and state, build UI separately, implement payment systems, and deploy and scale. They make building AI features easier, but you're still building a traditional application.

1. **LangChain/LlamaIndex:** These are libraries for building AI applications in code. They reduce boilerplate but still require developers to write Python or JavaScript code, manage execution infrastructure, handle persistence and state, build UI separately, implement payment systems, and deploy and scale. They make building AI features easier, but you're still building a traditional application.

2. **AutoGPT/BabyAGI:** These are autonomous Agent systems focused on goal-driven behavior. They're impressive demonstrations but require significant compute resources, have limited practical reliability, don't integrate with blockchain, lack monetization infrastructure, and are designed for research, not production applications.

3. **PersistentAI's Differentiation:**

- **No-code development** for most use cases
- **Durable execution** guarantees operations don't fail silently
- **Native blockchain integration** for payments and ownership
- **AG-UI protocol** for dynamic interfaces
- **Built-in monetization** through Agent NFTs
- **Telegram-first design** for maximum distribution

Deterministic Execution and Auditability

Security, Verification, and Trust: Given that PersistentAI Agents handle financial operations and sensitive user data, security and verifiability are paramount.

Every operation executed by a PersistentAI Agent is recorded with cryptographic integrity on TetraChain, a Layer 2 blockchain on TON. This creates an immutable audit trail that can be used for:

1. **Dispute Resolution:** If a user claims an Agent executed an unintended operation, the complete execution trace can be reviewed. Every decision point, every API call, every piece of data processed is recorded.

2. **Compliance:** Regulated operations (financial services, healthcare, etc.) can demonstrate compliance by providing auditable records of all Agent actions.

3. **Algorithm Transparency:** Users can inspect the decision-making process. If an Agent denied a loan application or made an investment recommendation, the reasoning is transparent and verifiable.

4. **Fraud Prevention:** Malicious Agents that attempt to deceive users leave evidence on-chain. Reputation systems can incorporate this data, and users can avoid Agents with suspicious behavior.

The Verifiable Execution Stack

PersistentAI's verification model has six layers:

1. **Flow Definition Hash:** The Agent's logic (the flow of connected nodes) is content-addressed. Any modification to the flow produces a different hash, preventing silent changes.

2. **Input Hash:** User inputs are canonically encoded (using CBOR format) to ensure identical inputs always produce identical hashes.

3. **DBOS Checkpoints:** Each node execution is persisted with state snapshots, enabling deterministic replay.

4. **Provider Attestations:** External services (LLM APIs, data feeds, oracles) sign their responses, creating cryptographic proof of what data was provided.

5. **State Root:** The final state of all variables is hashed into a Merkle tree, producing a compact state root.

6. **TetraChain Settlement:** Batch commitments are submitted to TetraChain L2 with optimistic finality (less than 3 seconds). A 24-hour dispute period allows anyone to submit fraud proofs if execution was invalid.

If a dispute arises, any party can:

1. Download the complete execution trace
2. Replay the Agent's operations locally
3. Compare their replay result with the on-chain commitment
4. Submit a fraud proof if discrepancies are found

This creates strong economic incentives for honest behavior: sequencers who submit invalid commitments are slashed, losing their stake.