An Intelligent Event-Driven Automation Framework for Decentralized Applications

Nagipragalathan N[1], Lin Eby Chandra J[2]
[1,2]*Department of Computer Science and Engineering Jaya Engineering College, Anna University, Chennai, India*

***Abstract*—Web3 gives people a way to own data, money, and digital assets without a central authority, but turning an idea into a working decentralized application is still out of reach for almost everyone. It normally takes smart contract programming, careful key handling, low level calls to blockchain nodes, and a separately built web interface. The result is that the space of useful Web3 ideas is far larger than the small group of engineers able to build them. This paper proposes AgentFlow Web3, a no-code and AI-assisted way to compose, run, and host decentralized applications. The core idea is simple: make every on chain action a building block that anyone can drop onto a visual canvas, wire together with logic and artificial intelligence, attach to a drag and drop interface, and publish at a shareable link, all without writing code. Because the building blocks are general, the same canvas can express a very wide range of applications, from decentralized finance automation and trading to non fungible token workflows, decentralized organizations, payments, on chain monitoring, identity, and autonomous agents that read and act on chain state. We describe the concept, a conceptual architecture, a taxonomy of composable primitives, and a broad set of application domains, and we report a proof of concept that implements the paradigm and runs real on chain workflows. Measured results show that the orchestration layer adds only a few milliseconds and that end to end time is set by the blockchain endpoint rather than by the platform, which supports the feasibility of the approach.**

***Index Terms*—No-Code Development, Web3, Blockchain, Decentralized Applications, Large Language Models, AI Agents, Visual Programming, Smart Contracts, Workflow Automation, Composability.**

I. INTRODUCTION

Every major wave of computing became important only after ordinary people could take part in it. Spreadsheets opened computation to analysts, the web opened publishing to anyone with a browser, and no-code tools now let non programmers build business software by connecting blocks on a screen. Web3 has not had that moment yet. The technology is powerful, with user owned assets, programmable money, and verifiable ownership, but building on it is still an expert activity.

A typical decentralized application asks one person to play several roles at once. They must write and audit smart contracts, manage wallets and private keys, talk to blockchain nodes through low level interfaces, and design and deploy a web interface that ties all of this together. Few people hold all of these skills, so most Web3 ideas never get built. The people who understand a real problem, such as a community organizer, a small business owner, an artist, or an analyst, are usually not the people who can ship a contract and a front end.

The same problem in ordinary software was solved by raising the level of abstraction. Visual automation platforms let a user express logic by connecting configurable blocks, and studies report that low-code and no-code technology now accounts for a large and growing share of new enterprise software [1], [2]. Language models have added a second lever, producing agents that can plan and carry out multi step tasks [3], [4], [5]. Neither lever has reached Web3 in a usable form. Mainstream automation tools expose no native blockchain actions, and most research that joins language models with blockchains

aims at auditing contracts or coordinating machines rather than at helping a person build.

This paper proposes a way to close that gap. We present AgentFlow Web3, a no-code and AI-assisted paradigm in which on chain actions are first class building blocks on a visual canvas. A user composes a workflow by connecting blocks, attaches it to a drag and drop interface, and publishes the result at a shareable link, with an assistant available throughout. Figure 1 shows the path from an idea to a hosted application. Because the building blocks are general purpose, the same surface can express a very wide space of applications rather than a single fixed product, which is the heart of the proposal.

Figure 1. From an idea to a hosted decentralized application, with no code.

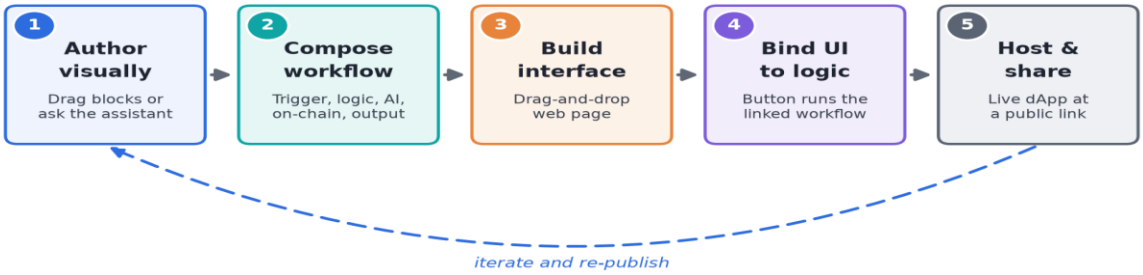

Figure 1. From an idea to a hosted decentralized application, with no code.

Our contributions are as follows.

- A paradigm that unifies visual workflow automation, native on chain actions, language model agents, and a drag and drop interface builder with built in hosting, so that the full path from logic to a shareable product happens in one place and without code.
- A taxonomy of composable on chain primitives, run by a topological engine through typed input and output handles, so that blockchain reads and writes combine as freely as ordinary logic and artificial intelligence.
- A broad map of application domains that the paradigm enables, from decentralized finance and trading to creator tools, organizations, payments, monitoring, identity, and autonomous agents, showing that the approach addresses a wide space of uses rather than one product.
- A proof of concept that implements the paradigm and runs real on chain workflows, with measured latency that isolates the negligible cost of the engine from the cost of the network and supports the feasibility of the design.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 presents the concept and architecture together with the taxonomy of primitives. Section 4 maps the application domains the paradigm enables. Section 5 reports a proof of concept and measured feasibility. Section 6 discusses the approach, and Section 7 concludes with future work.

II. RELATED WORK

2.1 No-Code and Low-Code Automation

Visual automation has grown from a convenience into a mainstream way of building software. Node-graph tools let people express logic by connecting configurable blocks, and current industry analyses report that low-code and no-code technology underpins a large share of new applications, with workflow automation among the leading uses [1]. A clear recent trend is the blending of automation with artificial intelligence, which turns fixed pipelines into systems that can decide at run time [2]. AgentFlow Web3 keeps the familiar node-graph style of tools such as n8n [6] but differs in one important way: it makes on chain actions native blocks rather than leaving them to generic web request nodes

2.2 Language Model Agents and Workflow Orchestration

Language models have driven a wave of work on agents that reason and act over several steps. The ReAct method showed that interleaving reasoning with actions lets a model build and revise a plan while gathering information from outside sources [3].

Tool use and multi agent coordination followed, with frameworks for self taught tool calling [7] and for conversation among cooperating agents [8]. Surveys have organized the field, covering the building blocks of autonomous agents [4] and the progress and open problems of multi agent systems [5], while benchmarks such as FlowBench study how explicit workflow knowledge improves agent planning and report that visual, flowchart style structure helps models most [9]. AgentFlow Web3 turns these ideas into a product by letting the user compose an explicit graph in which agent blocks sit beside deterministic on chain and logic blocks.

2.3 Blockchain Building Blocks and Accessible Web3

The Web3 stack rests on well defined standards. A general purpose contract platform provides the execution layer [10], [11], and standards such as the fungible token [12] and the non fungible token [13] define the assets that most applications touch. Oracle networks bring outside data, including price feeds, on chain [14], and account abstraction points toward smart accounts that remove the rigid private key model and improve usability for ordinary users [15]. Most work here targets contract correctness, oracle design, or machine to machine coordination. This paper takes a different angle and focuses on the human building experience, giving non experts a no-code surface that uses these same standards.

2.4 Visual Programming and No-Code Web Building

Node-based visual programming, where a user connects nodes to set a processing order, has long served automation builders, data pipelines, and chatbot designers, and open libraries for building such editors are now common [16]. Separately, website builders have moved from purely template based editing toward assisted generation of layouts and content [17]. AgentFlow Web3 draws on both lines and joins them, so the visual interface drives the visual logic and can be hosted and shared from inside the same platform.

III. THE AGENTFLOW WEB3 PARADIGM

The paradigm rests on three ideas. First, an on chain action should be a building block, not a programming task. A user who wants to read a balance, move a token, sign a message, or react to a contract event should place a block and fill in a few fields, in the same way they would add a step to any visual automation. Second, these blocks should compose freely with logic and with artificial intelligence, so that a workflow can branch on a condition, summarize data with a language model, and then act on chain, all in one ordered flow. Third, logic alone is not a product, so the platform also lets a user build an interface, bind it to the workflow, and host it at a shareable link. Together these form the loop in Figure 1: author, compose, build, bind, host, and share, then iterate.

3.1 Conceptual Architecture

The platform is organized as the layers in Figure 2. A visual authoring layer hosts the workflow canvas, the interface builder, and an assistant. An orchestration layer compiles a workflow into a dependency graph and runs its blocks in order. A capabilities layer provides two families of power, namely on chain primitives and language model agents. A composition and hosting layer binds an interface to a workflow and serves the result at a public link. A state layer stores the workflows, pages, runs, and settings. The layers talk through clear interfaces, so each can change without disturbing the others, and new primitives can be added without touching the rest of the system.

Figure 2. Conceptual platform architecture.

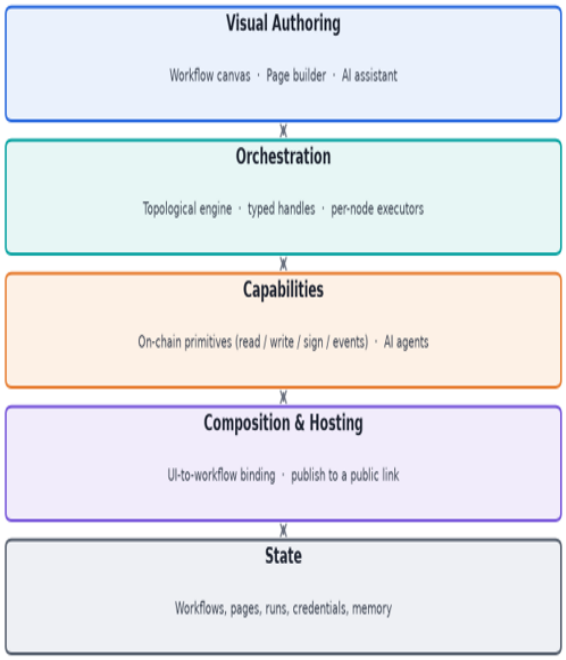

Figure 2. Conceptual platform architecture.

3.2 Composable Primitives

The expressive power of the paradigm comes from a library of typed blocks that combine on a canvas. Trigger blocks start a workflow by hand, on a schedule, from a webhook, or from a chat message. Logic and data blocks branch, filter, and transform. Agent blocks call language models for reasoning, extraction, or natural language replies, with memory and tools. Output blocks return a response or render a result. The defining family is the set of on chain primitives in Table 1, which makes blockchain actions native to any workflow. The set is representative rather than fixed, since the same pattern extends to further standards and networks.

Table 1. Representative on-chain primitives available as building blocks.

Primitive	On-chain function
Wallet	Provide an account and signer to later blocks
Read Balance / Token	Read native or token balances and metadata
Read Contract	Call a view function on any contract
Write Contract	Send a state changing contract call
Transfer (native / token / NFT)	Move value or assets
Sign Message	Produce a cryptographic signature
Event Trigger	React to recent contract events
Price Feed	Read oracle price data
Chain State	Read gas, blocks, and transaction status
Name Service	Resolve human readable names to addresses

3.3 Execution Model

A workflow is a set of blocks and directed connections. To run it, the engine builds a dependency graph, computes a topological order, and rejects any graph with a cycle or an unreachable block. Blocks then run in order, so a block executes only after the blocks feeding it have finished. The engine gathers each block's inputs from the outputs of upstream blocks, matching the typed handle on each connection, which lets a wallet block feed a transfer block on a dedicated input while the main flow continues on its own path. Because deterministic on chain blocks, conditional logic, and language model agents all run inside this one ordered pass, the same model expresses a simple balance check and a multi step automation that reasons and then acts on chain.

3.4 Interface Binding and Hosting

A workflow becomes a usable product through the interface builder and a binding step. A user assembles a page from drag and drop blocks and marks an action element, such as a button, as bound to a workflow. When the page is used, a small embedded script gathers any field values, runs the bound workflow, and shows the result in place. On publishing, the page is served at a public per-page link, and the platform keeps the binding with the page so the application stays wired to its logic even after the visual editor rewrites the markup. The outcome is a shareable application produced entirely through visual building.

3.5 Assisted Authoring

An assistant supports the user throughout. It answers questions about blocks and workflows in plain language, keeps a short conversation history, and routes requests to a chosen model backend. It supports building but does not hide the workflow, which stays fully visible and editable, so the user keeps an inspectable plan rather than an opaque result. This matters when the actions are real financial transactions.

IV. APPLICATION DOMAINS

Because the primitives are general, the paradigm is not tied to one product. The same canvas expresses a wide space of applications, a sample of which appears in Figure 3. Each domain reuses the same building blocks in a different arrangement, which is the source of the breadth.

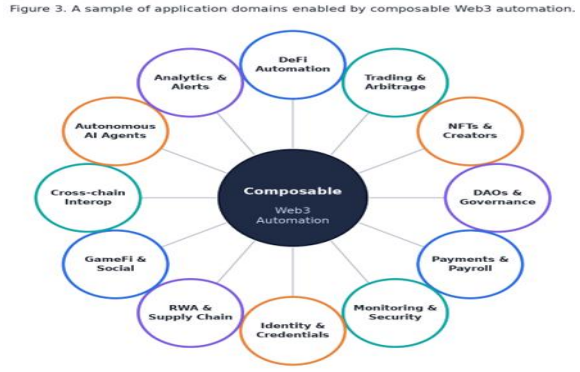

Figure 3. A sample of application domains enabled by composable Web3 automation.

Table 2 makes the breadth concrete by pairing each domain with one example automation and the primitives it would use. The examples are illustrative, not exhaustive, and most real applications combine several domains. A single workflow can, for instance, watch a price feed, ask an agent to judge a condition, move funds, and then update a hosted dashboard, crossing the finance, agent, and interface domains at once.

Table 2. Example automations across application domains.

Domain	Example automation	Primitives used
DeFi automation	Rebalance a position when a price threshold is crossed	Price feed, balance, contract write
Trading and arbitrage	Execute a swap when a price condition holds	Price feed, condition, contract write
NFTs and creators	Mint and split royalties when a sale event fires	Event trigger, write, transfer
DAOs and governance	Release treasury funds on an approved proposal	Event trigger, condition, transfer
Payments and payroll	Pay a list of contributors on a schedule	Schedule, wallet, batch transfers
Monitoring and security	Alert when a watched wallet moves funds	Event trigger, AI summary, output
Identity and access	Gate content by verifying on chain ownership	Balance or ownership read, condition
Real-world assets	Update a tokenized asset from oracle data	Oracle or web data, contract write
Autonomous AI agents	Read on chain state, reason, and act	AI agent, on chain read and write, memory
Analytics and alerts	Summarize on chain activity and notify	Read or event, AI summary, output

V. PROOF OF CONCEPT AND FEASIBILITY

To show that the paradigm is practical, we built a working platform that implements it and used the platform to run real on chain workflows. The proof of concept is a single web application that contains the authoring surfaces, the orchestration engine, and the on chain and language model services, with a managed database for storage. On chain work uses a standard Ethereum client library, and language model blocks use a vendor neutral interface so that several model providers are interchangeable. This section reports functional validation, measured latency, and a comparison with existing tools.

5.1 Functional Validation

Eleven on chain workflows covering reads, writes, signing, event queries, and price feeds were executed end to end. Balance and token reads returned correct holdings; native and token transfers produced valid transaction hashes, and a combined send and confirm workflow read back a successful receipt; a generic contract read returned a contract value; message signing returned a valid signature; an event workflow returned recent logs; a name service workflow resolved a known name to its address; and block and gas workflows returned current chain state. Every workflow finished with all of its blocks executed in the correct order. The same workflows were also driven from a hosted interface, where a button ran its bound workflow and showed the on chain result in place, which validates the full path from a built interface to the chain.

5.2 Feasibility: Does the No-Code Layer Cost Speed?

A fair question for any no-code paradigm is whether the convenience of building visually comes at the price of speed. We answer it directly by measuring the overhead the paradigm adds on top of raw blockchain access. Each workflow was run seven times and its execution time was recorded, spanning the whole graph from trigger to final block. Table 3 reports the median and range, and Figure 4 plots the medians on a logarithmic scale.

The measurements separate the two costs that make up any on chain automation: the cost of the orchestration itself and the cost of reaching the blockchain. Workflows run against a local chain, where the network cost is close to zero, finish in roughly four to twenty six milliseconds, a figure that includes a five block workflow that writes a transaction and then reads its receipt. Because this figure is essentially the engine plus a nearby node, it isolates the orchestration overhead the paradigm introduces, which is only a few milliseconds per workflow. Workflows that call a public network take about an order of magnitude longer, near two

hundred seventy milliseconds, and that gap is entirely the round trip to a public node rather than any work the engine performs.

The reading that matters for the idea is therefore positive: the no-code abstraction is effectively free. A builder pays almost nothing for composing on a visual canvas instead of writing code, and the achievable speed of an automation is set by the blockchain endpoint it talks to, not by the platform. Raising the level of abstraction so that anyone can build does not make the resulting applications slow, which is exactly the property the paradigm needs in order to be practical at the scale of its many use cases.

Table 3. Measured execution latency over seven runs per workflow.

Workflow	Blocks	Network	Median (ms)	Range (ms)
Sign Message	4	Local	4	4 to 6
Latest Block	3	Local	5	4 to 6
Read Token Supply	3	Local	6	6 to 8
Wallet Balance	3	Local	7	6 to 12
Token Transfer Events	3	Local	7	6 to 12
Token Balance	3	Local	12	10 to 18
Send and Confirm	5	Local	26	23 to 57
Gas Price	3	Public RPC	267	260 to 479
Price Feed	3	Public RPC	267	264 to 441

d

Figure 2. Measured per-workflow execution latency.

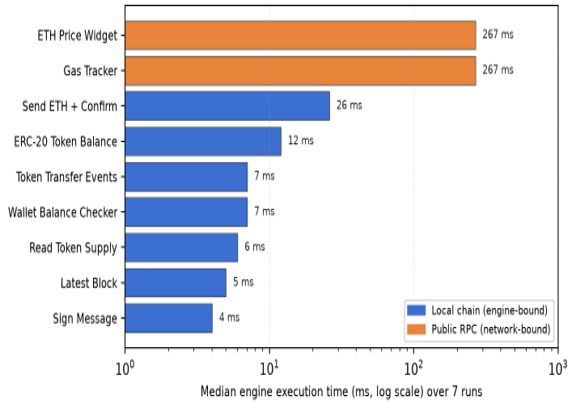

Figure 4. Median engine execution time per workflow, separating local chain (engine bound) from public network (network bound) workflows.

5.3 Capability Comparison

Table 4 compares the paradigm with four representative tools that a builder would otherwise have to combine: a general automation platform, a consumer automation service, a Web3 developer toolkit with hosting, and a contract development workflow built from a framework and a library. The comparison is qualitative and reflects native, out of the box capability rather than what could be scripted with enough effort. No single existing tool covers the full path from a visual on chain workflow to a bound, hosted, shareable interface, which is the gap this work addresses.

Table 4. Native capability comparison (Yes / Partial / No).

Capability	General automation	Consumer automation	Web3 toolkit	Contract framework	This work
Visual node-graph automation	Yes	Partial	No	No	Yes
No programming required	Yes	Yes	Partial	No	Yes
First-class on-chain read / write	No	No	Partial	Yes (code)	Yes
On-chain signing and events	No	No	Partial	Yes (code)	Yes
Built-in AI agent blocks	Partial	Partial	No	No	Yes
Integrated UI builder	No	No	Partial	No	Yes
UI bound to an on-chain workflow	No	No	Partial	No	Yes
One-click hosting of the app	No	No	Yes	No	Yes

VI. DISCUSSION

Three points stand out. First, treating on chain actions as native blocks, rather than as generic web requests, removes the constant switching that fragments normal Web3 work. A builder who reads a balance, calls a contract, signs a message, and then shows the result to a user does the whole loop on one canvas, which is the same kind of consolidation that no-code tools brought to business software [1], now applied to Web3.

Second, the breadth shown in Section 4 is not an accident of feature count. It follows from composability. A small set of general primitives, combined freely on a canvas, expresses many applications because each domain is a different arrangement of the same blocks. This is why the approach scales across uses rather than solving one problem, and it is the property that makes the space of supported ideas effectively open ended.

Third, the measured latency shows where the real cost lies. The engine adds only single digit milliseconds even for a multi block workflow that writes and confirms a transaction, while a single public network call costs about two orders of magnitude more. For a no-code tool this is the right shape, since the orchestration layer is effectively free and the achievable speed is set by the chain rather than by the platform. It also explains why a local chain is valuable during building, since it removes the network cost while a user iterates.

The approach has limits that we state plainly. On chain writes currently rely on a key supplied to a wallet block, which suits building and dedicated hot wallets but is not the keyless, recoverable model that account abstraction now offers to end users [15]. A block targets one network per run, so a cross network workflow must be expressed step by step. The evaluation measured functional correctness and latency rather than behavior under heavy load, and it did not yet include a controlled study with human builders, which would be the strongest test of the usability claim. These points set the agenda for future work.

VII. CONCLUSION AND FUTURE WORK

7.1 Conclusion

This paper proposed AgentFlow Web3, a no-code and AI-assisted paradigm for composing, running, and hosting decentralized applications. The central idea is to make every on chain action a building block that combines freely with logic, with artificial intelligence, and with a drag and drop interface, so that the full path from an idea to a shareable product happens on one canvas and without code. Because the building blocks are general, the same surface expresses a wide space of applications, from finance and trading to creator tools, organizations, payments, monitoring, identity, and autonomous agents. A proof of concept implements the paradigm, runs real on chain workflows, and shows that the orchestration layer adds only a few milliseconds while the network sets the end to end cost. Taken together, the concept and the evidence support a simple claim: with the right abstraction, building on Web3 can become as accessible as building an ordinary automation.

7.2 Future Work

Several directions follow from the limits above.

- Add account abstraction so that end users transact with keyless, recoverable smart accounts and sponsored fees, which removes key handling from the building experience and improves usability.
- Grow the assistant from question answering toward turning a described goal into an editable workflow, building on workflow guided planning for language model agents.
- Support multi network and Layer 2 composition so that one workflow can span chains and use lower fees and higher throughput.
- Run a controlled study with human builders to measure how quickly non experts produce a working application, and test the platform under concurrent load.
- Open a shared marketplace where builders publish, find, and remix workflow and interface bundles, which would turn the breadth of domains into a growing public library.

Declarations

Funding

No funding was received for conducting this study.

Conflicts of Interest

The authors declare no conflict of interest.

Data Availability

The AgentFlow Web3 source code and deployment configuration are publicly available at the project repository:.

Author Contributions

Nagipragalathan N: Conceptualisation, system design, implementation, deployment, writing. J. Lin Eby Chandra: Supervision, methodology validation, manuscript review and editing.

Use of Generative AI

The authors used a generative AI assistant to support language refinement, structural organisation, and proofreading of the manuscript. All technical content, system design, implementation, testing, and conclusions are the original work of the authors. AI generated suggestions were reviewed and verified by the authors, who retain full intellectual responsibility for the manuscript.

ACKNOWLEDGEMENTS

The authors thank the Department of Computer Science and Engineering, Jaya Engineering College, for the support extended throughout this project.

REFERENCES

[1] Kissflow, "No-Code Automation Benchmarks: 2025 to 2026 Enterprise Performance Data," 2025. [Online]. Available: https://kissflow.com/no-code/no-code-automation-benchmarks-enterprise-stats/

[2] ToolJet, "Low-Code AI Workflow Automation Tools for Modern Engineering Teams," 2025. [Online]. Available: https://blog.tooljet.com/low-code-ai-workflow-automation-tools/

[3] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," in Proc. Int. Conf. on Learning Representations (ICLR), 2023. arXiv:2210.03629.

[4] L. Wang, C. Ma, X. Feng, et al., "A Survey on Large Language Model based Autonomous Agents," Frontiers of Computer Science, vol. 18, no. 6, art. 186345, 2024. arXiv:2308.11432.

[5] T. Guo, X. Chen, Y. Wang, et al., "Large Language Model based Multi-Agents: A Survey of Progress and Challenges," in Proc. 33rd Int. Joint Conf. on Artificial Intelligence (IJCAI), 2024. arXiv:2402.01680.

[6] n8n GmbH, "n8n: Secure Workflow Automation for Technical Teams," 2025. [Online]. Available: https://n8n.io

[7] T. Schick, J. Dwivedi-Yu, R. Dessi, et al., "Toolformer: Language Models Can Teach Themselves to Use Tools," in Advances in Neural Information Processing Systems (NeurIPS), 2023. arXiv:2302.04761.

[8] Q. Wu, G. Bansal, J. Zhang, et al., "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation," 2023. arXiv:2308.08155.

[9] R. Xiao, W. Ma, K. Wang, et al., "FlowBench: Revisiting and Benchmarking Workflow-Guided Planning for LLM-based Agents," 2024. arXiv:2406.14884.

[10] V. Buterin, "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform," Ethereum White Paper, 2014. [Online]. Available: https://ethereum.org/en/whitepaper/

[11] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," Ethereum Yellow Paper, 2024 revision. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[12] F. Vogelsteller and V. Buterin, "EIP-20: Token Standard," Ethereum Improvement Proposals, 2015. [Online]. Available: https://eips.ethereum.org/EIPS/eip-20

[13] W. Entriken, D. Shirley, J. Evans, and N. Sachs, "EIP-721: Non-Fungible Token Standard," Ethereum Improvement Proposals, 2018. [Online]. Available: https://eips.ethereum.org/EIPS/eip-721

[14] L. Breidenbach, C. Cachin, B. Chan, et al., "Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks," Chainlink Labs, 2021. [Online]. Available: https://chain.link/whitepaper

[15] V. Buterin, Y. Weiss, K. Gazso, et al., "ERC-4337: Account Abstraction Using Alt Mempool," Ethereum Improvement Proposals, 2024. [Online]. Available: https://eips.ethereum.org/EIPS/eip-4337

[16] xyflow, "React Flow: A Library for Building Node-Based Editors and Interactive Diagrams," 2025. [Online]. Available: https://reactflow.dev
[17] GrapesJS, "GrapesJS: Open-Source Web Builder Framework," 2025. [Online]. Available: https://grapesjs.com
[18] wevm, "viem: A TypeScript Interface for Ethereum," 2025. [Online]. Available: https://viem.sh