Uzo: Measuring and Closing the Developer Gap on BOT Chain

The Uzo Labs Team
uzolabs.xyz · uzolabsxyz@gmail.com
Technical paper, v1.0, November 2026

Abstract

A new EVM chain can ship fast blocks, a block explorer, a faucet, a decentralized exchange and a bridge, and still be hard to build on. The pieces exist, but the path between them does not: network settings, contract addresses and tooling behaviour are scattered, partly undocumented and sometimes misdescribed. We study this gap on BOT Chain, a Parlia-based EVM layer 1, and present Uzo, an independent, open source toolkit that closes it.

We make three contributions. First, a measurement study of BOT Chain's developer surface: RPC behaviour, fee market, confirmation latency, contract verification, token standards and node availability. Several widely repeated assumptions turned out to be wrong, including the claim that `eth_getLogs` is disabled on the public mainnet RPC. Second, the design of Uzo Deploy, a browser deployment tool whose factories have no owner, hold no funds, give every user a deterministic address that nobody else can claim, submit every instance for verification automatically and charge a fixed fee that is forwarded within the same transaction. We state its security properties and the tests that support them. Third, the architecture of the wider toolkit: a typed SDK, starter templates, documentation, a reusable hackathon kit and a planned data service. Uzo Deploy has been live on BOT Chain mainnet since November 9, 2026.

1 Introduction

Developers judge a chain by how long it takes to go from an empty folder to a deployed, verified contract that a real user can call. On established chains that path is paved: libraries ship the chain definitions, explorers verify contracts automatically, indexers answer event queries and templates cover the common cases. On a new chain each of those steps is a small research project.

BOT Chain is a useful case. It is an EVM-compatible layer 1 with chain ID 677 on mainnet and 968 on testnet, an average block time of about 0.67 seconds, a Blockscout explorer (BOTScan), a Uniswap-style exchange (BDEX), a USDT bridge and a public faucet. On November 10, 2026, BOTScan reported 29.8 million blocks, 26.9 million transactions and 1.37 million addresses. The chain is used. Yet when we set out to build on it, we found:

- no published chain definition in common client libraries, so every project hand-writes one;
- contract addresses spread across several documentation pages, with no machine-readable source;
- documentation, including the operator's own, that described some behaviour differently from what we measured, most importantly the availability of event queries (Section 3);
- verification and token behaviour that differs from Ethereum in ways that break common tools.

None of these is a protocol flaw. Each one costs a new developer hours, and together they decide whether a hackathon team ships or gives up. This paper documents what we measured, how we built around it and what remains open.

Contributions.

- A measurement study of BOT Chain's developer surface, with every figure dated and reproducible (Section 3).
- The design and security argument of Uzo Deploy, a non-custodial, ownerless factory system for deploying and verifying contracts from a browser (Sections 4 and 5).
- The architecture of the Uzo toolkit and the design of a planned data service, Uzo Index (Sections 6 and 7).
- An evaluation of cost and latency on live networks, and an honest list of limitations (Sections 8 and 10).

Independence. Uzo Labs is an independent project and is not affiliated with or endorsed by BOT Chain. All measurements were taken against public endpoints. All Uzo code described here is open source under the MIT licence, except the planned hosted services.

2 Background and Model

BOT Chain. BOT Chain runs Parlia, a proof of staked authority consensus with fast finality, in a client whose public source is a fork of BNB Smart Chain's client at version 1.5.13 [1, 2]. Both public RPC endpoints identify as

`Geth/v1.5.13`. The chain supports the Cancun opcodes, including transient storage [3]. Its native token is BOT (18 decimals). USDT on BOT Chain has 6 decimals.

Actors. We consider four actors. A *deployer* uses Uzo Deploy to create a contract. A *user* interacts with that contract, for example by sending a tip. *Uzo* operates the website, the verification service and the fee wallet. An *adversary* may control any number of accounts, observe the mempool and front-run transactions.

Trust assumptions. We assume the chain is live and executes transactions correctly, and that the public RPC returns correct data. We treat BOTScan as honest but unreliable: it may be slow or unavailable, and the system must not depend on it for safety. Crucially, we treat Uzo itself as a party to be minimized: if Uzo's servers or keys are compromised, the attacker must not be able to take funds held by, or change the behaviour of, any contract a deployer created.

3 Measuring BOT Chain's Developer Surface

We measured the developer surface of both networks between October 3 and November 10, 2026, using the public RPC endpoints, BOTScan's APIs and transactions from a dedicated testnet key. Table 1 summarizes the findings that changed our design.

3.1 Fee market and latency

On both networks the base fee is always zero, and `eth_gasPrice` returns 20 gwei. Over a 50-block `eth_feeHistory` window, the 10th, 50th and 90th percentile rewards were 20, 20 and 22 gwei in 49 of 50 blocks (Oct 6). Every testnet transaction we sent paid exactly 20 gwei, so one million gas costs 0.02 BOT. `eth_estimateGas` overestimated actual usage by 1.5%, small enough that we show the estimate directly.

From signing in a wallet to a confirmed receipt took under about 2.5 seconds in our tests: sending took 0.8 seconds (2.2 seconds on a cold first call) and the receipt arrived 0.4 to 1.7 seconds later, polling every half second. On a chain this fast, a user interface can treat confirmation as nearly immediate, provided it polls rather than waits for a subscription.

3.2 Event queries

The most consequential finding was that event queries work. BOT Chain's own JSON-RPC documentation lists `eth_getLogs` as disabled on the public mainnet RPC, and we repeated that in our own early documentation. In our measurements it was not disabled for address-filtered queries. What is true is narrower: a single query that spans the full chain history for a busy contract exceeds the endpoint's 30-second limit. The practical rule is to always filter by address and to query in bounded ranges from a known starting block. Because the operator's documentation still lists the method as disabled, it could be switched off at any time; applications should keep an explorer-based fallback, and our own materials are being corrected.

3.3 Contract verification

BOTScan is a Blockscout deployment and supports standard JSON input, multipart and flattened source. Verification through its v2 API is asynchronous and completed in about 5 to 10 seconds for every instance we submitted. Verifying a factory with Foundry took 16 seconds. Resubmitting a verified contract returns an "already verified" response, so the operation is idempotent. Because explorer twin-matching does not cover contracts with per-instance immutables, any system that creates contracts on users' behalf must verify each instance itself.

3.4 Tokens and test funds

USDT has 6 decimals, which is a frequent source of off-by-10^{12} errors in code written for 18-decimal tokens. On testnet, the official faucet dispenses 10 tBOT per address per day and, separately, 1,000 test USDT per claim per day. Both testnet and mainnet USDT are admin-mintable contracts; on testnet a single address holds the minter and admin roles. We describe the token as "USDT on BOT Chain" and make no claim about its issuer.

4 Uzo Deploy: Design

Uzo Deploy lets a deployer pick a template, fill in a short form, sign one transaction and receive a verified contract with a page to manage it. Version 1 offers three templates: a fixed-supply ERC-20 token, an ERC-721 collection and a tip jar for BOT and USDT tips.

4.1 Goals

- **G1** **No custody.** No Uzo contract ever holds user funds, even transiently across transactions.
- **G2** **No admin keys.** Uzo has no role in any contract it helps create, and no factory has an owner, a pause or an upgrade path.
- **G3** **Predictable, unclaimable addresses.** A deployer sees their contract's address before signing, and no one else can take that address.
- **G4** **Verified by default.** Every instance is submitted to BOTScan for verification without the deployer doing anything, with a retry if it fails.
- **G5** **A bounded, honest fee.** The fee is fixed, shown before signing, never held and never charged for a failed deploy.

Table 1: Assumptions we started with, what we measured, and the design consequence.

Assumption	Measured (date)	Consequence
`eth_getLogs` is disabled on the public mainnet RPC	It works. Address-filtered queries over 100,000-block ranges return in seconds. A genesis-to-latest query for a busy contract (mainnet USDT) times out after 30 s with error −32002 (Oct 6).	Read events from the RPC in chunks, starting at each contract's deploy block. Use the explorer only as a fallback.
A WebSocket endpoint exists	None is published. An upgrade request returns HTTP 200, not 101 (Oct 6).	Poll for new blocks and receipts.
The explorer verifies identical bytecode automatically	Twin matching uses deployed bytecode, which includes immutables, so per-user instances never match. An exact twin is only linked and stays unverified. Verifying a factory does not verify its children (Oct 6).	Submit every instance for verification individually.
USDT supports EIP-2612 permit	It does not, on either network. Both are OpenZeppelin `ERC20`, `ERC20Burnable` and `AccessControl` contracts with no `DOMAIN_SEPARATOR` or `nonces` (Oct 6).	USDT payments take two transactions: approve, then pay.
Standard toolchains verify on BOTScan	Foundry gives a full match. Hardhat 3 prefixes source names with `project/`, which changes the metadata hash and therefore the bytecode of any factory that embeds a template (Oct 6).	Build and verify with Foundry only.
Anyone can run a node	The public client source contains a chain configuration for 968 but none for 677, its only genesis file is for an unrelated chain ID (698), and its bootnodes are private addresses (Oct 3).	Independent nodes are not possible from public materials. Build on the public RPC.
A protocol paymaster is available	We found no confirmed paymaster on chain 677 (Oct 3).	Do not promise gasless transactions. Use ERC-2771 meta-transactions with a relayer where needed [7].

4.2 Factories

Each template has one factory per version. A factory exposes two main functions. `deploy` creates an instance with CREATE2 [4] and emits

```
event Deployed(address indexed deployer,
  address indexed instance,
  bytes32 indexed templateId, uint16 version);
```

and `predictAddress` returns the address that `deploy` will produce for the same inputs. The salt is

$$s = \text{keccak256}(\text{abi.encode}(msg.sender, userSalt)).$$

Because the caller's address is part of the salt, a front-runner who copies a pending transaction gets a different address, so no one can deploy to an address predicted for someone else (G3). Each template also implements `uzoTemplate()`, returning its identifier and version, so tools can recognize an instance from the chain alone.

Factories are themselves deployed through the widely used deterministic CREATE2 deployer [8] with a fixed salt. Their addresses therefore depend only on their bytecode and constructor arguments. The token and NFT factories have the same address on mainnet and testnet. The tip jar factory differs between networks because its constructor takes the network's USDT address.

4.3 The fee

Every factory inherits a small fee component with two immutable values set in its constructor: `fee` and `feeRecipient`. There is no setter and no owner, so neither can change. `deploy` first checks that the call carries exactly `fee` (otherwise it reverts with `WrongFee`), then creates the instance and emits `Deployed`, and only as its final step forwards the fee to `feeRecipient`. If any step reverts, the whole transaction reverts and no fee is taken. The fee is 0.1 BOT on mainnet and 0.1 tBOT on testnet. Tips never pay it.

4.4 Templates

Token. An ERC-20 with EIP-2612 permit [5] and burning, built from OpenZeppelin Contracts v5.7.0 [9]. The constructor mints the entire supply to one recipient. There is no owner and no further mint path.

NFT collection. An ERC-721 with an immutable maximum supply, owner-only `mint` and `mintBatch`, ERC-2981 royalties capped at 10% [6], two-step ownership transfer and a one-way `freezeMetadata` switch. The owner is whoever the deployer names; Uzo has no role. `mintBatch` updates the minted count before its safe-mint loop, so a reentrant `onERC721Received` callback cannot exceed the cap.

Tip jar. A contract that forwards tips to one immutable recipient. `tipNative` forwards the full `msg.value` in the same call. `tipToken` moves USDT directly from the tipper to the recipient with `transferFrom`, so the tokens never pass through the jar. The jar has no `receive` or `fallback` function, so plain transfers to it revert. Both tip functions are guarded by a transient-storage reentrancy lock, emit their event before the external call, and cap messages at 140 bytes and titles at 64 bytes to bound log costs. The USDT address is fixed per factory, and the deployment script refuses to continue unless it matches the address published in the Uzo SDK.

4.5 Automatic verification

After a deploy, the browser asks Uzo's verification service to verify the new address. The service never accepts source code or constructor arguments from the client. Instead it:

1. reads the creation receipt from the RPC and finds a `Deployed` event from a factory listed in Uzo's own configuration, rejecting any other address;
2. checks that `uzoTemplate()` on the instance matches the event;
3. rebuilds the constructor arguments from the factory call;
4. submits the standard JSON input from Uzo's own build to BOTScan; and
5. polls until the explorer reports the contract as verified.

The standard JSON inputs are checked in continuous integration, so they cannot drift from the deployed bytecode.

4.6 Reading data without a dedicated indexer

Pages that list a deployer's contracts or a jar's tips read `Deployed` and `Tipped` events. Following Section 3, the primary source is chunked, address-filtered `eth_getLogs` against the public RPC, starting from each factory's recorded deploy block. BOTScan's logs API is the fallback. Both sit behind one interface, so either can be replaced.

5 Security Properties

We state the properties Uzo Deploy relies on and the evidence for each. The contracts are tested, not audited (Section 10).

Property 1 (No custody). No function of a factory or tip jar leaves funds in it: every amount received through those functions is forwarded in the same transaction.

Argument. Factories forward the exact fee in the same call that receives it. Tip jars forward native tips in the same call and never receive USDT, because `transferFrom` moves it from tipper to recipient directly. With no `receive` or `fallback`, plain BOT transfers revert. Tokens sent with a direct ERC-20 transfer, or BOT forced in by SELFDESTRUCT, cannot be prevented by any contract; they would be stuck, since nothing can withdraw them, but no user funds are ever routed that way. *Evidence:* three invariants on the tip jar (native balance zero, token balance zero, recipient received everything) over 256 runs of depth 64, and unit tests on each factory.

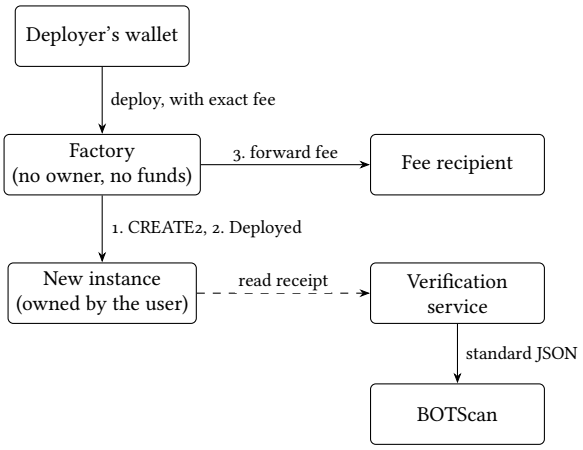


Figure 1: A deploy. The factory creates the instance, emits `Deployed` and forwards the fee last, all in one transaction. Verification happens afterwards and never affects the contract.

Property 2 (No admin keys). No address controlled by Uzo can change, pause, upgrade or withdraw from any factory or instance.

Argument. Factories have no owner, no setters and no proxy. Their only state is immutable. Token and tip jar instances have no owner at all. NFT instances have exactly one owner, chosen by the deployer at creation. This holds even if Uzo's servers or the fee wallet are compromised: the fee wallet only ever receives.

Property 3 (Unclaimable addresses). For any deployer d and salt u, only d can create a contract at `predictAddress`$(d, u, \ldots)$.

Argument. The CREATE2 salt includes `msg.sender`. *Evidence:* fuzz tests confirm that `predictAddress` matches `deploy` for random inputs (10,000 runs per factory) and that a different caller yields a different address.

Property 4 (Fixed supply and capped collections). A token's total supply never increases after construction, and an NFT collection never exceeds its maximum supply.

Evidence: unit and fuzz tests, including a reentrancy test through `onERC721Received` on `mint`, which shares the batch minting path.

Test and analysis summary. The contract suite has 98 test functions: unit, fuzz (10,000 runs), invariant and deployment-script tests. Continuous integration fails unless line, statement, branch and function coverage of the contract sources is 100%. Before the fee was added, Slither 0.11.6

Table 2: Components and status, November 2026.

Component	What it does	Status
SDK	Chain definitions, addresses, ABIs, explorer client and BDEX swaps	Early release (v0.3)
Templates	Clone-and-deploy starters	Partly released
Documentation	Guides and reference	Live
Uzo Deploy	Browser deploys with verification	Live on mainnet
Hackathon kit	Event kit for students and organizers	In development
Uzo Index	Decoded data and webhooks	Planned

[10] reported only four informational results across 102 detectors, each documented: one deliberate low-level call in `tipNative`, needed to forward BOT to contract recipients such as multisigs, and three false positives on embedded creation code. The fee component adds a second deliberate low-level call of the same kind, to forward the fee. Five fork tests run the tip jar against the real testnet USDT contract.

Off-chain risks. The contracts cannot protect a user from signing something else. A compromised website could ask a wallet to sign a malicious transaction. Uzo Deploy shows the factory address on the review step with a link to its verified source on BOTScan, sends headers that stop the site being framed or its content types being guessed, and keeps per-wallet and tip pages out of search engines so that the Uzo name does not lend credibility to tokens other people deploy. Uzo does not review or endorse contracts created with it.

6 The Uzo Toolkit

Uzo Deploy is one part of a toolkit whose aim is a single, verifiable path from an empty folder to a shipped application. Table 2 gives the status of each part at the time of writing.

SDK. The SDK (`@uzolabs/sdk`) publishes BOT Chain's chain definitions for viem [11], the addresses of WBOT, USDT, Multicall3, BDEX's Permit2, the BDEX routers and the bridge for both networks, the matching ABIs, a typed BOTScan client and, since version 0.3, a module that quotes BDEX swaps and builds unsigned swap calls with a 5% slippage cap. Its purpose is that no project ever re-types chain data. Uzo Deploy imports its network data from it.

Templates. Released templates cover a token, an NFT collection, a BDEX integration and a gasless application. Because no paymaster is confirmed on the chain, the gasless template uses ERC-2771 meta-transactions with a self-hosted relayer [7]. Bridge and AI-agent templates are in development.

Documentation. The documentation follows one rule: describe what is live, label everything else. Every page about a product that is not released carries a status notice, and every address is drawn from a single source.

Hackathon kit. A kit for running a BOT Chain hackathon, to be released under the MIT licence, designed so that any organizer can fork it and change one configuration file. It includes a starter application, track guides, workshops, a judging rubric and scripts that check, for each team, that a submitted contract exists on testnet, is verified and was deployed during the event. It is in development and will be tested in a student dry run before its first event.

7 Uzo Index: Design

Uzo Index is a planned service that turns raw chain data into ready-to-query datasets and webhooks. Its design follows directly from the measurement study.

Positioning. Because event queries work on the public RPC, raw logs are not scarce. What is scarce is decoded, aggregated data that no one computes for BOT Chain today: prices, candles and volume for BDEX pools, token holders, bridge transfer tracking and push notifications when a payment arrives. Uzo Index provides those, and does not try to replace general-purpose indexers [15, 14].

Finality first. Datasets are written only from finalized blocks, which removes reorganization handling from every table. Webhooks in the first version also fire only on finalized data. A payment application must never release goods on an event that could still be reverted.

Honest status for bridge transfers. The index sees only BOT Chain. It therefore reports what happened on BOT Chain and nothing more: a deposit has *left BOT Chain*, an arrival has *arrived on BOT Chain*, a refund was *refunded on BOT Chain*. It never reports a transfer as completed on another chain. Destination chain identifiers are mapped explicitly, including Tron mainnet (728126428) and Tron's Nile testnet (3448148188).

System transactions. Parlia's validator reward and fee-distribution calls are recorded separately and excluded from transfer datasets, activity feeds and webhooks, because they are not payments anyone made.

8 Evaluation

Cost. Table 3 gives the total gas of the three mainnet smoke-test deploys on November 9, 2026, read from their receipts, and the cost at the measured 20 gwei. Each transaction also paid the 0.1 BOT fee. At these prices the fee is between about 3.5 and 10.5 times the gas cost of a deploy.

For a tipper, a USDT tip costs an approval (46,961 gas measured for a new allowance) plus a transfer (58,254 gas mea-

Table 3: Gas of the mainnet smoke-test deploys and cost at 20 gwei.

Template	Gas	BOT
Token	949,287	0.0190
NFT collection	1,430,654	0.0286
Tip jar	477,756	0.0096
Deploy fee (all templates)		0.1000

sured for `transferFrom` to a new holder), a little over 0.002 BOT in total.

Latency. On testnet, the three smoke-test instances of the first factory versions (before the fee) were verified on BOTScan 9.8, 8.4 and 5.7 seconds after submission. Combined with a sign-to-confirm time of under about 2.5 seconds, a deployer sees a verified contract in roughly 10 to 15 seconds after signing.

Deployment. The first factory versions were deployed and verified on testnet on October 6, 2026. The current, fee-charging factories were deployed and verified on testnet and mainnet on November 9, 2026. Full addresses are listed in Appendix A.

9 Related Work

OpenZeppelin Contracts [9] supplies the audited building blocks our templates inherit, and its Wizard generates similar contracts for local deployment. Browser deployment platforms such as thirdweb offer prebuilt contracts across many chains; Uzo Deploy differs by having no owner on its factories, holding no funds and verifying every instance by default on a chain where explorer twin-matching does not apply. Sourcify [13] and Blockscout [12] provide source verification; our service automates the latter for factory-created contracts. Deterministic deployment with CREATE2 [4, 8] is standard; binding the salt to the caller to prevent address squatting is a common pattern that we adopt. For data, The Graph [14] and Ponder [15] index arbitrary contracts; Uzo Index instead offers fixed, decoded datasets specific to BOT Chain.

10 Limitations and Future Work

- **No audit yet.** The contracts are tested, statically analysed and documented, but have not had an independent audit or external review. We advise against using them for large amounts until they have.
- **Dependence on public infrastructure.** Uzo Deploy reads from the public RPC and verifies through BOTScan. Neither affects contract safety, but outages degrade the experience. Independent nodes require a public mainnet genesis and reachable peers.
- **Two-step USDT payments.** Without permit on USDT, every first USDT payment needs an approval. BDEX's own Permit2 deployments, which its Universal Routers use on both networks, are a candidate for a later version. These are not the canonical Permit2 address used on Ethereum; SDK 0.2.1 corrected this after we found that both routers embed BDEX's deployments.
- **Metadata hosting.** NFT metadata is the owner's responsibility. Freezing metadata stops changes to the base URI, not to the content that URI serves.
- **Planned work.** An agent vault template with on-chain spending limits, Uzo Index, the hackathon kit's first events and further templates.

11 Conclusion

BOT Chain has the pieces developers need. What it lacked was a reliable path between them, and some of what developers believed about the chain was simply out of date. By measuring the chain directly, correcting those beliefs and building tools whose guarantees are enforced by contracts rather than promised by operators, Uzo aims to make that path short, honest and verifiable.

References

[1] BOT Chain. Node client source. https://github.com/BOTChain-bot/BOTCHAIN. Inspected October 3, 2026.

[2] BNB Chain. BNB Smart Chain documentation: consensus and fast finality. https://docs.bnbchain.org.

[3] Ethereum Improvement Proposals. EIP-1153: Transient storage opcodes. https://eips.ethereum.org/EIPS/eip-1153.

[4] Ethereum Improvement Proposals. EIP-1014: Skinny CREATE2. https://eips.ethereum.org/EIPS/eip-1014.

[5] Ethereum Improvement Proposals. EIP-2612: Permit extension for EIP-20 signed approvals. https://eips.ethereum.org/EIPS/eip-2612.

[6] Ethereum Improvement Proposals. ERC-2981: NFT royalty standard. https://eips.ethereum.org/EIPS/eip-2981.

[7] Ethereum Improvement Proposals. ERC-2771: Secure protocol for native meta transactions. https://eips.ethereum.org/EIPS/eip-2771.

[8] Arachnid. Deterministic deployment proxy. https://github.com/Arachnid/deterministic-deployment-proxy.

[9] OpenZeppelin. OpenZeppelin Contracts, version 5. https://github.com/OpenZeppelin/openzeppelin-contracts.

[10] J. Feist, G. Grieco and A. Groce. Slither: A static analysis framework for smart contracts. In *WETSEB*, 2019.

[11] wevm. viem: TypeScript interface for Ethereum. https://viem.sh.

[12] Blockscout. Smart contract verification. https://docs.blockscout.com.

[13] Sourcify. Decentralized source code verification. https://sourcify.dev.

[14] The Graph. Indexing protocol documentation. https://thegraph.com/docs.

[15] Ponder. Open source framework for blockchain application backends. https://ponder.sh.

A Deployed Factory Addresses

All factories are verified on BOTScan. The live list is maintained at https://docs.uzolabs.xyz/deploy/contracts; always confirm an address there and on the explorer before signing.

Network	Factory	Address	Deploy block
Mainnet (677)	UzoTokenFactory	`0x9D66023a9ab4a062701f0432156a3AC2Fb7964F6`	29,697,532
Mainnet (677)	UzoNFTFactory	`0xaF987D1BaEaAAcfD74b347618119AdeC04a21814`	29,697,533
Mainnet (677)	UzoTipJarFactory	`0x4925f24c3f51A31F1Dd4EbFE744b3e265240261c`	29,697,534
Testnet (968)	UzoTokenFactory	`0x9D66023a9ab4a062701f0432156a3AC2Fb7964F6`	29,861,516
Testnet (968)	UzoNFTFactory	`0xaF987D1BaEaAAcfD74b347618119AdeC04a21814`	29,861,516
Testnet (968)	UzoTipJarFactory	`0x95cB60ad9cB39fCF9085250F9A49BD8B04753a9C`	29,861,516

Disclaimer. This paper describes software and measurements as of November 2026. It is not financial or legal advice. Uzo Labs is an independent project and is not affiliated with or endorsed by BOT Chain. All product names are the property of their owners.