YYSFold System Walkthrough

A Complete Guide to the Dashboard and Pipeline

YYSFold Team

December 5, 2025

Abstract

This document provides a comprehensive walkthrough of the YYSFold blockchain fingerprinting system. It explains what you see on the live dashboard, how each component works, and the complete data pipeline from raw blockchain data to behavioral fingerprints.

Contents

1 What Is YYSFold?

YYSFold is a **blockchain behavioral fingerprinting system**. It watches blockchain activity in real-time, compresses each block into a compact "fingerprint," identifies unusual patterns (hotzones), and can predict what the next block might look like—all while generating cryptographic proofs that the analysis was done correctly.

Live Demo: `https://yysfold.ngrok.io`

2 Dashboard Overview

2.1 Header and Navigation

The header contains:

- **YYSFold logo** — Links to home
- **Atlas** — 2D/3D visualization of the vector space
- **Chat** — AI assistant for querying block data

2.2 Latest Block Fingerprint Card

The main card shows the most recent block fingerprint with:

Element	**Description**
sol · #383106448	Chain (Solana) and block height
[Live]	Real-time SSE connection active
Updated 7s	Time since last heartbeat
Tags (pills)	Semantic behaviors detected in this block
Block hash	Truncated block identifier
[raw block]	Link to download the raw block JSON
[proof]	Link to the ZK proof artifact
Regime	Dominant behavioral pattern combination
Anomaly: 0.37	Score from 0–1 (higher = more unusual)
Density	Peak hotzone density (KDE value)
PQ	Average PQ reconstruction error

2.2.1 Anomaly Labels

Score Range	**Label**	**Meaning**
< 0.45	Typical	Normal activity
$0.45 - 0.75$	Unusual	Notable patterns
> 0.75	Rare	Potential anomaly

2.3 Predicted Next Block Card

Shows predictions for the next block:

- **Chain** — Which blockchain (SOL, ETH, AVAX)
- **Confidence** — How sure we are (e.g., 24%)

- **Predicted Tags** — Expected behaviors
- **ETA** — Estimated time until next block
- **Reason** — Why we made this prediction

How predictions work:

1. Keep a rolling history of recent mempool snapshots
2. Compute trends: gas slope, transaction delta, tag frequency changes
3. If trends are stable → MIXED_ACTIVITY with low confidence
4. If gas spiking + DEX activity → HIGH_FEE + DEX_ACTIVITY

2.4 Mempool Ticker (Live Feed)

Shows real-time pending transactions:

- **Timestamp** — When the snapshot was taken
- **Chain** — Which blockchain
- **Pressure** — High / Normal / Low
- **Stats** — Pending tx count, gas price, total value
- **Highlights** — Notable patterns detected

Important: This shows the **mempool** (pending transactions), NOT confirmed blocks.

2.5 Block Detail Page

When you click on a block:

- **Hotzones** — Up to 16 density clusters, each with tags
- **Hypergraph** — 3D graph of hotzone relationships
- **Metrics** — Transaction count, PQ residuals, commitment hash

2.6 Atlas Page

Shows all hotzones from recent blocks projected into 2D or 3D:

- **Position** — Coordinates from dimensionality reduction
- **Size** — Proportional to density
- **Color** — Mapped from dominant semantic tag
- **Edges** — Hyperedges connecting related hotzones

3 The Pipeline

3.1 Step 1: Block Ingestion

```
RPC Node --> watchIngest.js --> Raw Block JSON
```

The ingestion worker polls blockchain RPC endpoints:

- **Ethereum** — via ETH_RPC_URLS
- **Solana** — via Solana RPC
- **Avalanche** — via AVAX RPC

Downloads: block header, transactions, execution traces, state changes.

3.2 Step 2: Vectorization

```
Raw Block --> vectorizeBlock() --> Vectors
```

Each transaction becomes a **16-dimensional vector**:

```
Transaction {amount, fee, gas, ...}
    --> Vector [0.72, 0.15, 0.33, 0.5, ...]
```

Encoded features: normalized amounts, fees, gas, contract types, assets, addresses, timestamps.

3.3 Step 3: Folding

```
[1000+ vectors] --> foldVectors() --> [6 vectors]
```

Compress all vectors into 6 summary vectors:

1. **Mean vector** — Average of all transactions
2. **Variance vector** — How spread out values are
3. **Components 1–4** — Principal directions of variation

Reduces 1000 transactions from ∼16,000 numbers to just 96.

3.4 Step 4: Product Quantization (PQ)

```
[6 vectors] --> pqEncode() --> [6 codes] + residuals
```

Each vector compressed using a codebook:

```
Vector [0.72, 0.15, 0.33, 0.5]
    --> Split into 4 subvectors
    --> Find nearest centroid for each
    --> Code: [142, 87, 203, 56]
```

The **residual** measures compression error.

3.5 Step 5: Hotzone Detection (KDE)

```
PQ codes --> pqDecode() --> Vectors --> KDE --> Hotzones
```

Run Kernel Density Estimation:

```
For each vector:
    density = sum of Gaussian kernels

If density > threshold:
    This is a HOTZONE
```

Hotzones get semantic tags based on vector component values.

3.6 Step 6: Hypergraph Construction

```
Hotzones --> buildHypergraph() --> Graph
```

Connect hotzones that are close in vector space and both high density.

3.7 Step 7: Anomaly Scoring

```
(density, residuals, tags) --> computeAnomalyScore() --> 0.37
```

Component	**Weight**	**Measures**
Density	50%	Activity concentration
Residuals	35%	PQ compression quality
Tags	15%	Suspicious tag presence

3.8 Step 8: Cryptographic Commitment

```
Fingerprint --> BLAKE3 hash --> Commitment
```

Create tamper-proof commitments for folded vectors, PQ codes, and codebook.

3.9 Step 9: ZK Proof Generation

```
Witness + Public Inputs --> Halo2 Prover --> Proof
```

Generate zero-knowledge proof that:

1. Vectorization was correct
2. Folding math is right
3. PQ encoding respects error bound
4. Commitments match

Current status: Running mock proofs. Real proofs need compiled Halo2 binaries.

3.10 Step 10: Storage and Display

```
Summary + Proof --> SQLite + JSON --> Dashboard
```

Saved to SQLite (for queries) and JSON files (raw artifacts).

4 Services Architecture

4.1 Local Development

Service	Port	Purpose
Next.js Dashboard	3000	Web UI
ngrok tunnel	—	Public URL
Ingest Worker	—	Fetches blocks
Mempool Worker	—	Watches pending txs

4.2 Cloud Deployment

Service	Platform	Purpose
API	Render	Serves data
Ingest Worker	Render	Block ingestion
Mempool Worker	Render	Mempool watching
Dashboard	Vercel	Next.js frontend

5 Key Files

File	Purpose
`scripts/ingestBlocks.ts`	Main ingestion pipeline
`scripts/mempoolWatch.ts`	Mempool polling + predictions
`folding/vectorize.ts`	Transaction to Vector
`folding/fold.ts`	Vectors to Folded Block
`folding/pq.ts`	Product Quantization
`analytics/hotzones.ts`	KDE + hotzone detection
`analytics/hypergraph.ts`	Graph construction
`zk/halo2Backend.ts`	ZK proof interface
`dashboard/app/page.tsx`	Main dashboard page

6 Quick Explanation Script

"YYSFold takes raw blockchain blocks and compresses them into behavioral fingerprints. Each block with thousands of transactions gets reduced to just 6 summary vectors using statistical folding and product quantization.

We then run kernel density estimation to find 'hotzones'—clusters of similar transaction patterns—and tag them semantically (DEX activity, bridge transfers, potential AML alerts, etc.).

The anomaly score tells us if a block looks normal or unusual compared to baseline. The mempool feed shows pending transactions in real-time, and we use trend analysis to predict what the next block might look like.

Everything is cryptographically committed with BLAKE3 hashes, and we can optionally generate zero-knowledge proofs that the analysis was done correctly—so you can verify the fingerprint without re-running all the math yourself."

7 Glossary

Term	Definition
Fingerprint	Compact representation of a block's behavior
Folding	Compressing many vectors into few summary vectors
PQ	Product Quantization—vector compression using codebook
KDE	Kernel Density Estimation—finding dense regions
Hotzone	A cluster of similar transactions
Hypergraph	Graph where edges can connect 2+ nodes
Anomaly Score	0–1 measure of how unusual a block is
Mempool	Pool of pending (unconfirmed) transactions
ZK Proof	Cryptographic proof of correct computation
Commitment	Hash that binds data without revealing it
SSE	Server-Sent Events—real-time data streaming
RPC	Remote Procedure Call—API to blockchain nodes