Gensyn Testnet Data Analysis Report

Project Summary

This project explores transaction-level data on the Gensyn Testnet to uncover key activity patterns, smart contract usage, gas efficiency, and anomalies. The analysis uses Python (Pandas, NumPy, Matplotlib, Scikit-learn) and is visualized with an interactive Streamlit dashboard.

Section 1: Network Overview and Address Activity

What did you do?

- Loaded and parsed the flattened Gensyn testnet transaction dataset.
- Identified the top 10 sender addresses by transaction count and ETH volume.
- Identified the top 10 recipient addresses by transaction count.
- Measured smart contract interaction frequency.
- Counted unique active sender addresses.
- Calculated block-level congestion ratios.

Why did you do this?
To determine whether the network shows organic usage or controlled behavior and to establish a foundational understanding of address-level activity and testnet saturation.

How did you do it?
Used groupby, filtering, and basic aggregation in Pandas. Calculated congestion using **gas_used_block / gas_limit_block**.

What did you find?

- The top sender address made over 90,000 transactions.
- The top recipient address received over 675,000 transactions.
- Most top senders by volume transferred about 43–44 ETH.
- 99.99% of transactions were contract interactions.

- Only 80 unique sender addresses existed.
- Average congestion ratio was 0.1577, indicating low utilization.

Answer / Conclusion
The Gensyn testnet appears synthetic and centralized, with low address diversity and nearly all transactions directed to a few contracts. The data suggests stress-testing or simulation, not live usage. Congestion is minimal, indicating good scalability.

Section 2: Gas Price, Value, and Network Congestion

What did you do?

- Created a scatter plot of log-transformed gas price vs. ETH value.
- Calculated and visualized transaction-level and block-level congestion ratios.
- Computed and visualized gas cost in ETH for each transaction.

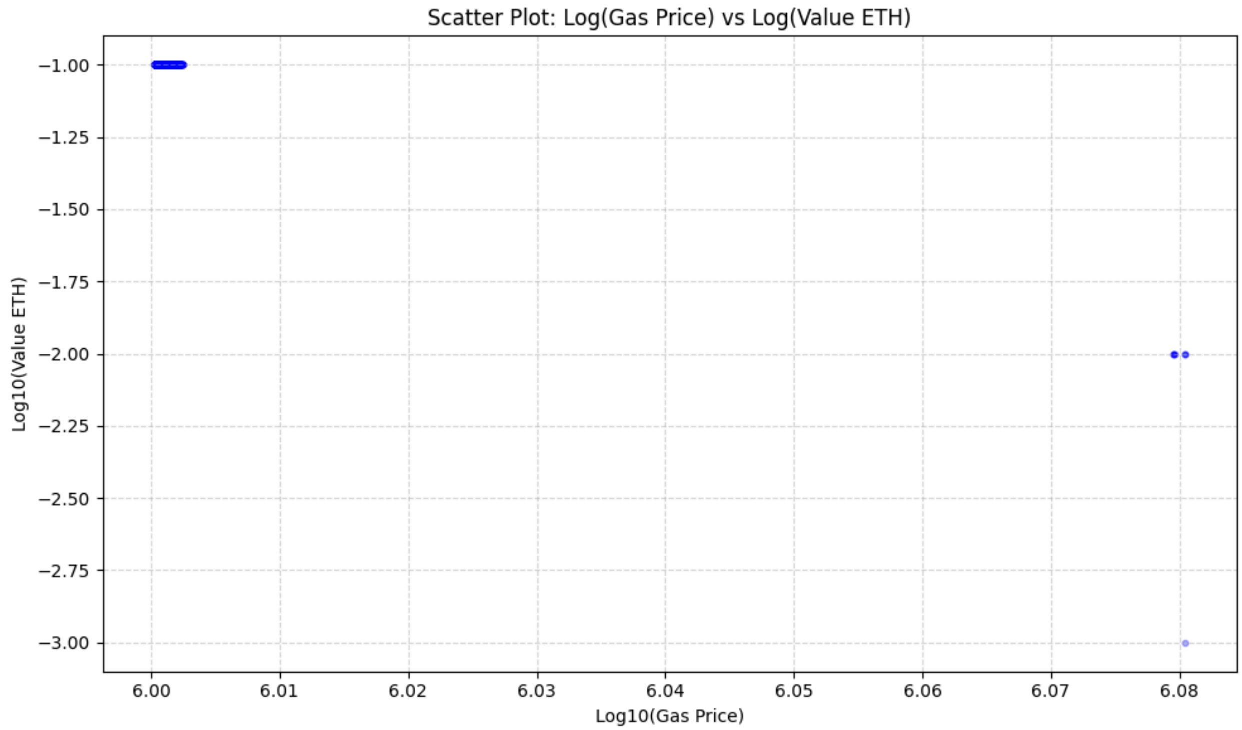
Why did you do this?
To evaluate the cost structure and capacity utilization of the network, assess efficiency, and understand the relationship between transaction cost and transferred value.

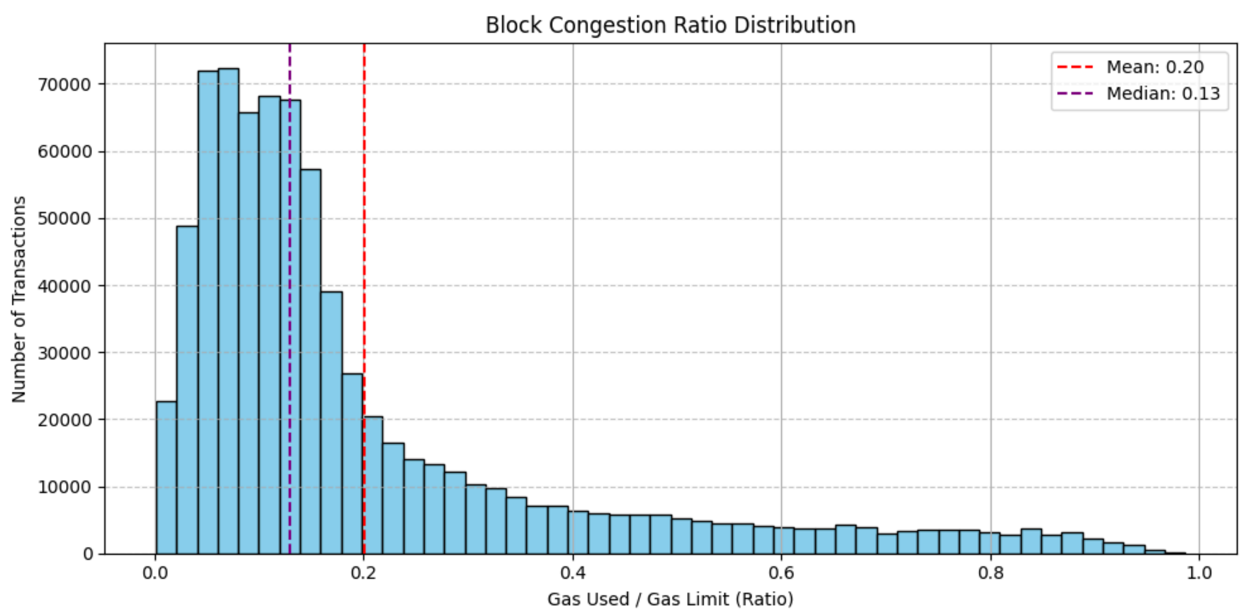
How did you do it?
Applied log transformation to skewed data. Calculated gas cost as `(gas_price * gas) / 1e18`. Used histograms and scatter plots for distribution analysis.

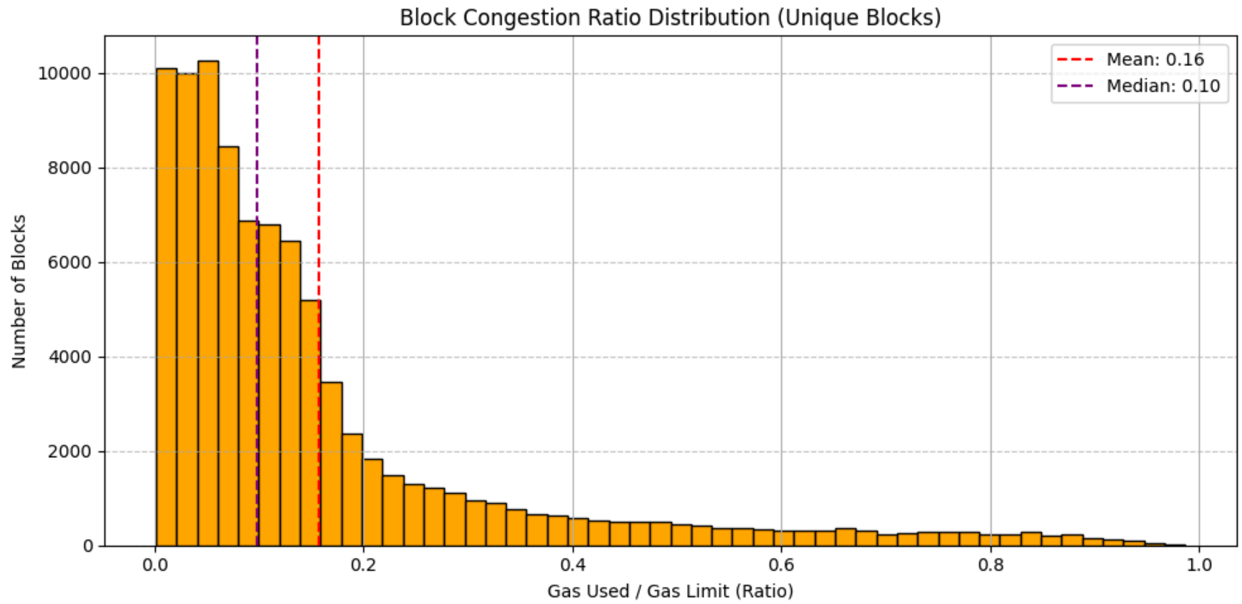
What did you find?

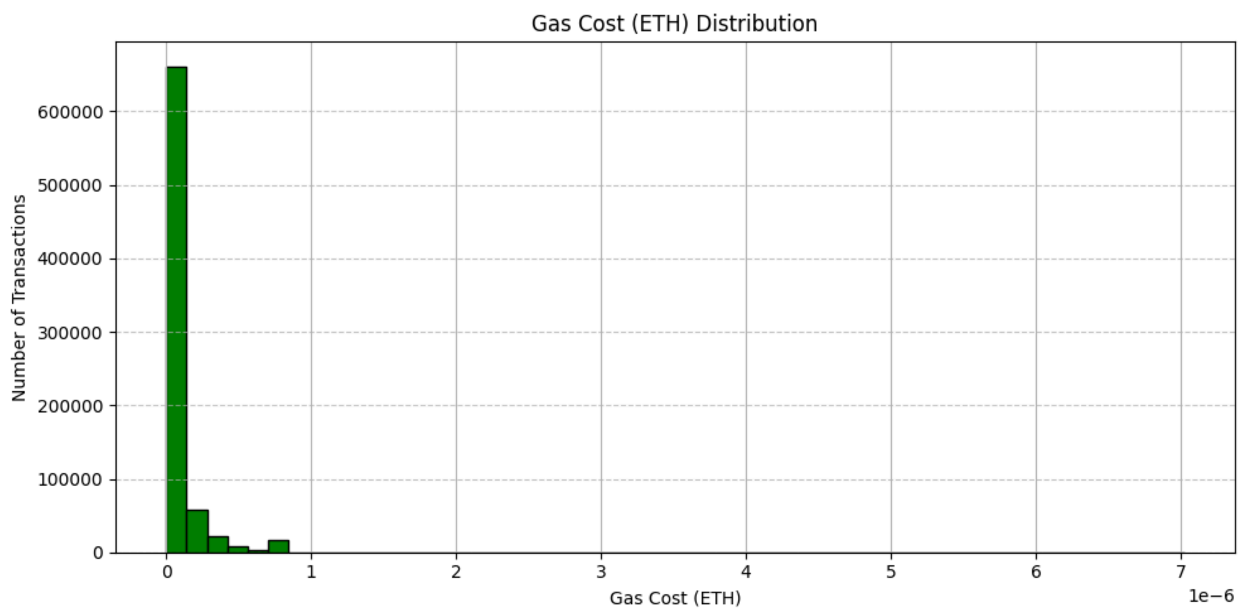
- Gas prices showed minimal variation.
- Most transactions transferred very small ETH amounts.
- Congestion was low across the board (mean: 0.1577, median: 0.10).
- Gas costs were tiny, with a mean of 8.49e-8 ETH and a max of 7.02e-6 ETH.

Answer / Conclusion
The network is highly cost-efficient and uncongested, which is ideal for testing. However, the lack of economic diversity and consistent pricing again suggests scripted or synthetic usage.


Scatter Plot: Log(Gas Price) vs Log(Value ETH)
Log10(Value ETH)
Log10(Gas Price)
−1.00
−1.25
−1.50
−1.75
−2.00
−2.25
−2.50
−2.75
−3.00
6.00
6.01
6.02
6.03
6.04
6.05
6.06
6.07
6.08


Block Congestion Ratio Distribution
Mean: 0.20
Median: 0.13
Number of Transactions
Gas Used / Gas Limit (Ratio)
0
10000
20000
30000
40000
50000
60000
70000
0.0
0.2
0.4
0.6
0.8
1.0





Section 3: Smart Contract Interactions and Gas Efficiency

What did you do?

- Filtered transactions with non-empty inputs to identify contract interactions.
- Identified top contracts by call volume and gas consumption.
- Calculated gas efficiency (value transferred per unit of gas spent).

- Measured input payload sizes and visualized their distribution.

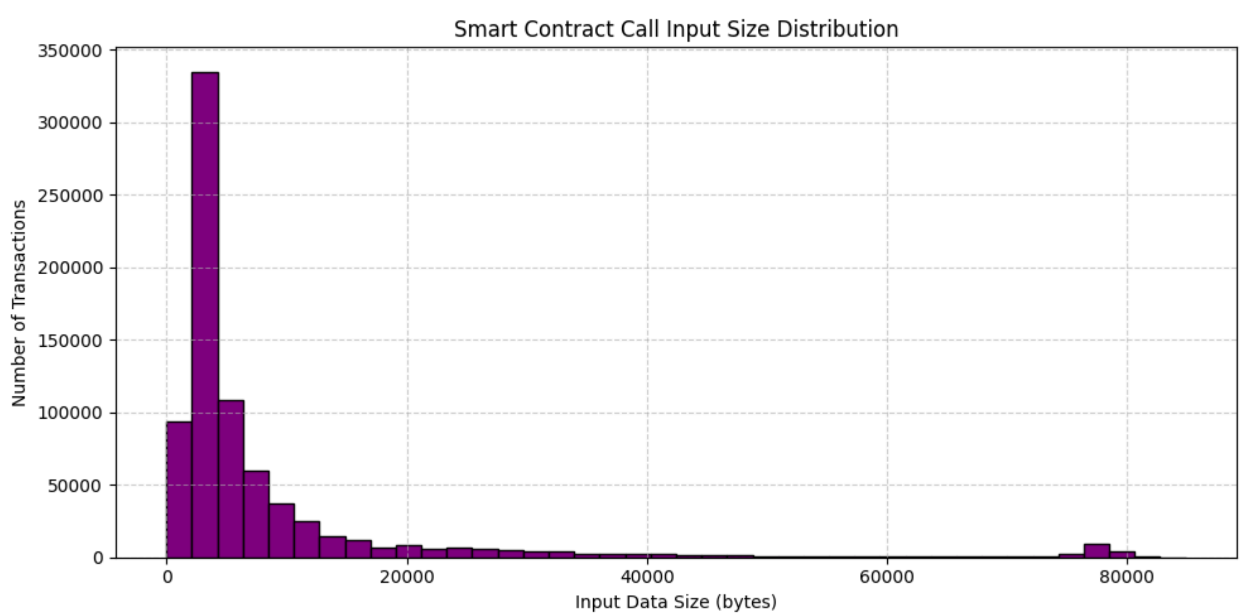
Why did you do this?
To identify dominant contracts, understand network compute behavior, and assess efficiency. Input size provides a proxy for complexity.

How did you do it?
Grouped data by recipient address. Computed gas expenditure and value-to-gas ratios. Measured input_size by string length.

What did you find?

- Two contracts accounted for nearly all contract interactions and gas usage.
- Top wallets paid ~0.004 ETH in gas, suggesting automated behavior.
- Some transactions had extremely high value-to-gas ratios (>1.5M).
- Input sizes were tightly clustered, implying repeated patterns or templates.

Answer / Conclusion
The testnet is dominated by two system-level contracts and scripted interactions. Gas use is extremely efficient, and transaction complexity is minimal. This supports the hypothesis of a controlled testing environment.



Section 4: Anomaly and Outlier Detection

What did you do?
Performed anomaly detection via:

1. **Z-Score Filtering** – flagged transactions where **$|z| > 3$**.
2. **Gas Inefficiency Ratio** – identified transactions with extremely high gas cost per unit of value.
3. **KMeans Clustering** – found outliers based on distance from cluster centroids after log transformation and scaling.

Why did you do this?
To detect unusual or inefficient behaviors, potential bugs, or edge-case usage. These could be important for debugging or abuse prevention.

How did you do it?
Used **zscore** from SciPy, domain-specific inefficiency ratios, and **KMeans(n_clusters=5)** with standardized, log-transformed features.

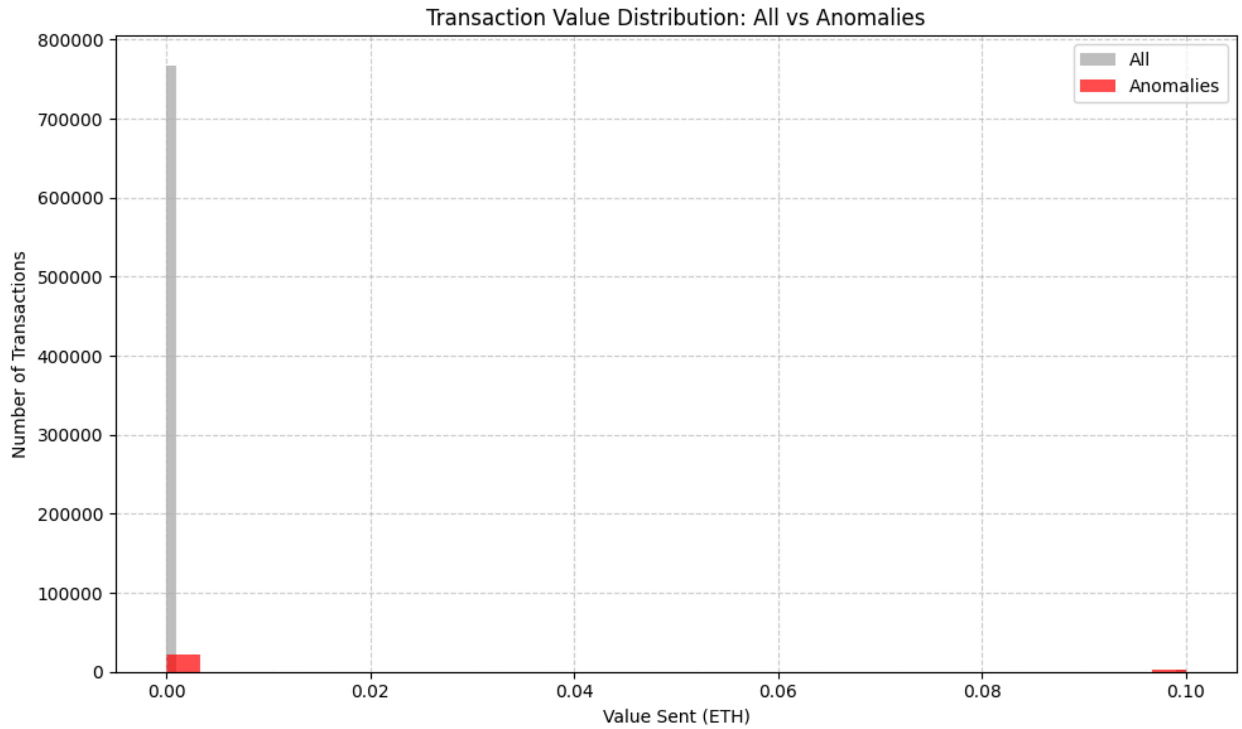
What did you find?

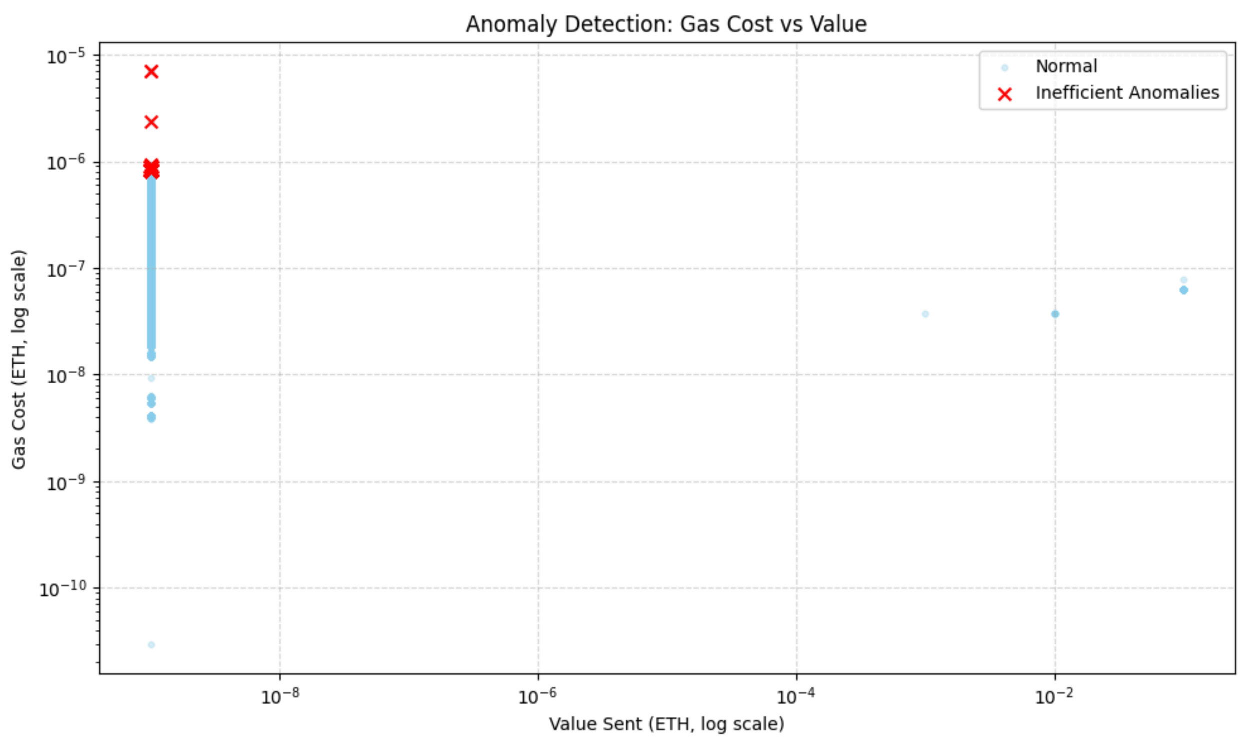
- 25,731 z-score anomalies
- 22,681 payload-heavy transactions
- 6,798 inefficient transactions
- 7,698 multi-feature outliers via clustering

Visualizations included:

- Log-log scatter plot of gas cost vs. value (with anomalies marked)
- Histograms comparing full vs. anomalous distributions

Answer / Conclusion
The testnet includes a substantial number of anomalous transactions — likely test scripts, bots, or stress scenarios. These methods are effective for identifying inefficiencies and may be even more powerful when applied to production usage.





Future Scope

The current analysis was based on a dataset spanning only two days of testnet activity. As a result, it was not possible to perform any meaningful time-series or temporal trend analysis.

With a longer observation window, future work could include:

- Hourly or daily transaction volume trends
- Peak usage periods and block-level throughput variation
- Gas price fluctuation over time
- Contract usage lifecycle or decay patterns
- User retention or repeated address analysis

Additionally, applying the current anomaly detection and efficiency metrics to a broader dataset would improve robustness and help distinguish between transient test behavior and structural inefficiencies.

Final Thoughts

This analysis shows that the Gensyn testnet is:

- Cost-efficient and low-congestion
- Dominated by a few actors and contracts
- Primarily used for stress testing, not organic interaction

The tools and metrics developed here can scale to production networks for real-time monitoring, optimization, and anomaly detection once live user data becomes available.
