Below is a **complete, self-contained plan** for a minimal but credible Proof-of-Concept (PoC) that demonstrates the two core breakthroughs:

1. The **transfer homomorphism delta circuit** (Δθ sub-circuit) – a lightweight Halo2 circuit that proves a delta vector correctly represents a private transfer.
2. A **minimal LatentLedger encoder simulation** that shows the homomorphism holds ($\mathcal{E}\theta(S_{t+1}) \approx \mathcal{E}\theta(S_t) + \delta(tx)$ with error < 1e-9) on 100,000 simulated transfers using toy states.

The PoC is deliberately scoped to be achievable in **8–12 weeks** by 1–2 engineers with ZK/ML experience.

PoC Scope Summary

- Toy state: maximum 128 accounts, balances quantized to 32-bit fixed-point (enough to show the property while keeping circuits small).
- Small transformer encoder: 6 layers, 8 heads, hidden dim 256 → embedding dim 512 (≈1.2M parameters, easily fits in Halo2).
- Train on synthetic transfer sequences until homomorphism error < 1e-9 on held-out data.
- Implement Δθ sub-circuit in Halo2 that computes the delta for a single transfer.
- Implement a verification circuit that proves prev_embedding + delta = new_embedding (with bounded error check).
- Run 100,000 simulated transfers offline: apply deltas additively, then occasionally run full encoder on the updated state and measure error.
- Produce a standalone demo binary + video showing:
    - Training convergence
    - Homomorphism error distribution (< 1e-9)
    - Halo2 proof generation/verification for a batch of deltas

Hardware & Software Requirements

Training the model

- 1 × NVIDIA RTX 4090 or A100 (24 GB VRAM) – sufficient for the 1.2M-parameter model.
- Alternative: 2 × RTX 4080/4090 if using DeepSpeed ZeRO-3.
- CPU: modern 16-core (Ryzen 9 or i9).
- RAM: 64 GB.
- Expected training time: 4–8 hours to reach < 1e-9 error.

Proving (Halo2)

- CPU-only proving is acceptable for PoC (no GPU needed for proving).

- Recommended: AMD Ryzen 9 7950X or Intel i9-13900K (32+ threads) → proving a single batch (256 txs) in ~8–15 seconds.
- Alternative: Apple M2 Ultra/M3 Ultra (proving ~12–20 seconds).
- RAM: 128 GB recommended for large constraint systems.

Software Stack

- Rust 1.75+ (nightly for Halo2 features)
- Halo2 (latest from privacy-scaling-explorations/halo2)
- EZKL 0.3+ (highly recommended – converts trained PyTorch/ONNX model to Halo2 circuit automatically)
- PyTorch 2.2+ with CUDA
- DeepSpeed (optional for faster training)
- Python 3.11 for training scripts
- Cargo-make or just for build orchestration

Epics, User Stories, Tasks & Sub-tasks

Epic 1 – Development Environment & Tooling Setup

Goal: All dependencies installed, reproducible builds, basic benchmarks.

Story ID	User Story	Acceptance Criteria
POC-ENV-01	As a developer, I can clone the repo and build all components in < 10 minutes on a fresh Ubuntu 22.04 machine	One-command setup script succeeds on clean VM
POC-ENV-02	As a developer, I can run a sanity-check Halo2 proof (e.g., simple range proof) in < 5 seconds	Proof generates and verifies successfully
POC-ENV-03	As a developer, I can train a dummy 6-layer transformer on random data in < 1 hour	Training script completes, loss decreases

Tasks

Task ID	Detailed Technical Task	Owner (example)	Acceptance / Deliverable	Est. Effort
ENV-01.01	Create monorepo structure with /training, /circuits, /demo	Lead Engineer	Git repo with README.md and cargo workspaces	1d

ENV-01.02	Write setup.sh script: install Rust nightly, CUDA 12, PyTorch, EZKL, Halo2 dependencies	DevOps	Runs on Ubuntu 22.04 and macOS without manual steps	2d
ENV-01.03	Add docker/Dockerfile for reproducible environment	DevOps	docker build . && docker run starts shell with all tools	2d
ENV-01.04	Benchmark basic Halo2 proving on target hardware	ZK Engineer	Table of proving times for 1M–8M constraint circuits	1d
ENV-01.05	Add CI pipeline (GitHub Actions) for lint, test, build	DevOps	Runs on push, fails on breakage	2d

Epic 2 – Synthetic Dataset & Toy State Simulation

Goal: Generate realistic transfer sequences for training and testing.

Story ID	User Story	Acceptance Criteria
POC-DATA-01	As a researcher, I can generate 500,000 transfer sequences with up to 128 accounts	Dataset saved as Parquet, reproducible with fixed seed
POC-DATA-02	As a developer, I can simulate state updates and compute exact deltas offline	Script produces CSV of before/after states and true deltas

Tasks

Task ID	Detailed Technical Task	Owner	Acceptance / Deliverable	Est. Effort
DATA-01.01	Define toy state format: Vec<(account_id: u32, balance: u128)>	ML Engineer	Rust struct + serialization	1d
DATA-01.02	Implement transfer generator: random sender/receiver/amount (1–10^9), ensure sufficient balance	ML Engineer	100% valid transfers, no overdrafts	2d
DATA-01.03	Generate training set (400k sequences) and validation set (100k sequences)	Data Engineer	Parquet files, total ~8 GB	1d

DATA-01.04	Write offline simulator: start from genesis state, apply transfers sequentially, record state snapshots every 100 txs	ML Engineer	CSV with state_hash, embedding (later filled)	2d

Epic 3 – Train Minimal Encoder & Demonstrate Homomorphism (Software Only)

Goal: Train a small transformer that achieves transfer homomorphism with < 1e-9 error.

Story ID	User Story	Acceptance Criteria
POC-TRAIN-01	As an ML researcher, I can train a 6-layer encoder until validation homomorphism error < 1e-9	Training log shows error curve, final checkpoint saved
POC-TRAIN-02	As a developer, I can run inference on toy states and measure additive error on 100,000 transfers	Script outputs error distribution (max < 1e-9)

Tasks

Task ID	Detailed Technical Task	Owner	Acceptance / Deliverable	Est. Effort
TRAIN-01.01	Implement 6-layer transformer encoder in PyTorch (rotary embeddings, GELU, 8 heads, hidden 256 → output 512)	ML Engineer	Model definition + random init test	3d
TRAIN-01.02	Define loss: reconstruction loss + explicit homomorphism regularisation term λ *		embed(S+tx) - (embed(S) + delta_approx)	
TRAIN-01.03	Train on RTX 4090/A100 with batch size 64, AdamW, lr 3e-4, cosine schedule	ML Engineer	Reach < 1e-9 error in ≤ 8 hours	4d (wall time)
TRAIN-01.04	Export final model to ONNX	ML Engineer	Verified ONNX file loads correctly	1d
TRAIN-01.05	Write evaluation script: load 100k validation transfers, compute additive error distribution	ML Engineer	CSV + plot showing max error < 1e-9	2d
TRAIN-01.06	Freeze best checkpoint as encoder_final.onnx	ML Engineer	Committed to repo	1d

Epic 4 – Delta Circuit (Δθ) & Homomorphic Update Proof in Halo2

Goal: Prove in zero-knowledge that a delta correctly updates the embedding.

Story ID	User Story	Acceptance Criteria
POC-DELTA-01	As a ZK engineer, I can generate a Halo2 proof that prev + delta = new with bounded error	Proof size ≤ 1 KB, verifies in < 100 ms on desktop
POC-DELTA-02	As a demo user, I can run a batch of 256 transfers and get a single aggregated proof	End-to-end script succeeds

Tasks

Task ID	Detailed Technical Task	Owner	Acceptance / Deliverable	Est. Effort
DELTA-01.01	Convert trained ONNX encoder to Halo2 circuit using EZKL	ZK-ML Engineer	ezkl gen-settings and ezkl compile-circuit succeed	3d
DELTA-01.02	Create lightweight Δθ sub-circuit: given sender_id, receiver_id, amount → delta vector (linear projection from learned embeddings)	ZK Engineer	Separate small circuit (~300k constraints)	4d
DELTA-01.03	Implement HomomorphicUpdate circuit: public inputs prev_embedding, delta, public output new_embedding; prove new = prev + delta and		new - encoder(full_state)	
DELTA-01.04	Generate trusted setup for the update circuit (MPC with 16 contributors or use public Halo2 params)	ZK Engineer	SRS file committed	2d
DELTA-01.05	Write proving script: load prev_embedding, compute delta offline, prove update	ZK Engineer	Proof < 1 KB, verify < 100 ms	3d
DELTA-01.06	Add batch aggregation (256 deltas → single proof) using recursion	ZK Engineer	Single proof for 256 tx batch	6d

Epic 5 – End-to-End Demo & Documentation

Goal: Polished demo that anyone can run and verify the claims.

Story ID	User Story	Acceptance Criteria
POC-DEMO-01	As an external reviewer, I can run the demo script and see homomorphism error < 1e-9 on 100k transfers	Script runs in < 30 min on recommended hardware
POC-DEMO-02	As a presenter, I have a video + slides showing the entire flow	5–8 minute video uploaded

Tasks

Task ID	Detailed Technical Task	Owner	Acceptance / Deliverable	Est. Effort
DEMO-01.01	Write master demo script: train → evaluate homomorphism → generate batch proof → verify	Lead Engineer	cargo run --release -- demo	4d
DEMO-01.02	Create README with hardware requirements, step-by-step run instructions	DevRel	Clear enough for external cryptographer to reproduce	2d
DEMO-01.03	Record 5–8 minute screencast: training curve, error histogram, proof generation/verification timings	Lead Engineer	Uploaded to YouTube/unlisted + repo	2d
DEMO-01.04	Write short technical summary (4–6 pages) explaining the PoC, limitations, next steps	Lead Engineer	PDF in repo	3d

To achieve realistic test data for training the 6-layer transformer (POC-ENV-03 and POC-TRAIN stories) and generating 500,000 transfer sequences (POC-DATA-01), start with **purely random/synthetic data** for quick pipeline validation (e.g., uniform random amounts and accounts). Then upgrade to **realistic data** derived from public Ethereum transactions for better homomorphism emergence (power-law activity, clustered transfers, log-normal amounts).

Real Ethereum data provides the most authentic patterns without privacy issues (public ledger).

Option 1: Quick Synthetic Data (For Initial Dummy Training, <1 Hour Setup)

Use this for POC-ENV-03 to verify the training script works fast.

Generate fully in code (Rust or Python) – no downloads needed.

Characteristics: Random sender/receiver, uniform amounts (1–10^9 wei), random initial balances.

Example Python Script (run locally with numpy/pandas):

Python

```python
import numpy as np
import pandas as pd
import random

NUM_ACCOUNTS = 128
NUM_SEQUENCES = 500000
MAX_TX_PER_SEQ = 100  # Vary sequence length

accounts = [f"0x{i:032x}" for i in range(NUM_ACCOUNTS)]
data = []

for seq_id in range(NUM_SEQUENCES):
    num_tx = random.randint(10, MAX_TX_PER_SEQ)
    for _ in range(num_tx):
        sender = random.choice(accounts)
        receiver = random.choice([a for a in accounts if a != sender])
        amount = random.randint(1, 10**9)  # Wei
        data.append({"seq_id": seq_id, "sender": sender, "receiver": receiver, "amount": amount})

df = pd.DataFrame(data)
df.to_parquet("synthetic_transfers.parquet")  # ~5-10 GB, compressible
```

Realism Level: Low (uniform, no clusters/power-law).
Time: <10 minutes to generate/run.

Use For: Dummy training to hit <1 hour goal.

Option 2: Realistic Data from Public Datasets (Recommended for Homomorphism Training)

Download real Ethereum transactions and subsample to 128 accounts.

Best Sources (free, downloadable CSVs/Parquet):

- **Kaggle Ethereum Transaction Dataset (RAW)**: ~1-2M recent transactions, full fields (from, to, value, block_number). Direct download. URL: https://www.kaggle.com/datasets/ikjotsingh221/ethereum-transaction-dataset-raw

- **Kaggle Ethereum Fraud Detection Dataset**: ~100k-500k txs, includes native ETH transfers + labels (bonus for realism). URL: https://www.kaggle.com/datasets/vagifa/ethereum-frauddetection-dataset
- **Kaggle BigQuery Ethereum Mirror**: Full historical dump (large, sample it). URL: https://www.kaggle.com/datasets/bigquery/ethereum-blockchain

Steps to Process (Python Example, Run Locally):

1. Download CSV → load with pandas.
2. Filter native ETH transfers: df = df[df['value'] > 0] (ignore contract calls if input_data not empty).
3. Sort by block_number/timestamp.

Build graph → extract subgraph:
Python

```
import networkx as nx
import pandas as pd

df = pd.read_csv("ethereum_transactions.csv")  # Your downloaded file
df = df[['from', 'to', 'value', 'block_number']].dropna()
df = df[df['value'] > 0]  # Native transfers only
df['value'] = pd.to_numeric(df['value'], errors='coerce')

G = nx.from_pandas_edgelist(df, source='from', target='to', edge_attr='value',
create_using=nx.MultiDiGraph())
# Find largest connected component or random dense subgraph
subgraph_nodes = max(nx.connected_components(G.to_undirected()), key=len)
subgraph_nodes = list(subgraph_nodes)[:128]  # Limit to 128 accounts

sub_df = df[df['from'].isin(subgraph_nodes) & df['to'].isin(subgraph_nodes)]
sub_df = sub_df.sort_values('block_number')

# Split into sequences (e.g., 500k chunks of 50-200 txs)
# Add initial balances by simulating from genesis (assume 0, or estimate from data)
```

4. sub_df.to_parquet("realistic_transfers_128accounts.parquet")
5. Generate sequences: chunk into 500k transfer groups for training batches.

Realism Level: High (real addresses, power-law degrees, bursty activity, real amounts).
Time: 1-2 hours (download + process).

Size: Subsampled ~500k-1M txs fits memory.

Option 3: Query Fresh Data from Google BigQuery (Most Up-to-Date)

Free public dataset: bigquery-public-data.crypto_ethereum.transactions

Sample Query (run in BigQuery console, export CSV/Parquet):

```
SQL
SELECT block_number, from_address, to_address, value
FROM `bigquery-public-data.crypto_ethereum.transactions`
WHERE value > 0  -- Native ETH transfers
  AND block_timestamp >= '2025-01-01'  -- Recent data
LIMIT 1000000  -- Adjust for size
ORDER BY block_number
```

Then subsample as in Option 2.

Realism Level: Highest (live, current patterns).

Time: 30-60 min + export.

Recommendations

- Start with Option 1 for dummy training (<1 hour total).
- Switch to Option 2 (Kaggle download) for realistic homomorphism results – real data helps the transformer learn linear directions faster.
- If you need >500k sequences, chain multiple Kaggle datasets or re-query BigQuery.
- Bonus: Add ERC20 transfers (e.g., USDT) from logs table for richer state (multiple assets).

This gets you production-like data while staying in the 128-account toy limit.

Total Estimated Effort

≈ 85–100 engineer-days (≈ 4–5 months calendar time for 1–2 engineers working full-time).

Next Steps After PoC

This PoC gives you:

- Concrete evidence that transfer homomorphism is achievable with small models.
- A working Halo2 delta/update circuit.
- A foundation to scale up to the full 24-layer, 7.9M-constraint LatentLedger.