How to Validate Rolv

100% Secure

• Zero IP Exposure

• Works on Any Hardware

You can now test ROLV on any CPU, GPU, or custom chip you own — laptop, on-prem cluster, cloud GPUs, phone, or ASIC.

Nothing proprietary is ever sent.

You only send us a small public JSON file with baseline results and hashes.

Why this is 100% secure

• The code you run contains zero ROLV IP.
• You generate the inputs and baseline on your own hardware.
• We run the full proprietary ROLV harness on the exact same inputs and return a professional comparison report.

Standard test (recommended, can be changed, see below)

• Matrix: 20,000 × 20,000
• Sparsity: 70%
• Batch size: 5,000
• Iterations: 1,000Step-by-step (takes < 3 minutes)

1. **Install dependencies (once):**

pip install torch numpy

2. **Download and run the ROLV Verification Kit (one file, works everywhere):**

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
#
# ROLV Verification Kit — Public baseline harness (ZERO ROLV IP)
# Run this on ANY hardware you own

import torch
import numpy as np
import hashlib
import json
import time
import random
import argparse
import platform
import os
from dataclasses import dataclass
from typing import Tuple, Dict, Any

@dataclass
class TestConfig:
    N: int = 20000
```

```
    zeros_pct: float = 0.70
    batch_size: int = 5000
    iters: int = 1000
    warmup: int = 20
    seed: int = 42
    pattern: str = "random"

def generate_matrix(cfg: TestConfig) -> Tuple[torch.Tensor, torch.Tensor]:
    torch.manual_seed(cfg.seed)
    np.random.seed(cfg.seed)
    random.seed(cfg.seed)

    density = 1.0 - cfg.zeros_pct
    A = torch.rand(cfg.N, cfg.N, dtype=torch.float32)

    if cfg.pattern == "random":
        mask = torch.rand(cfg.N, cfg.N) < density
        A *= mask.float()
    elif cfg.pattern == "power_law":
        vals = torch.rand(cfg.N * cfg.N)
        vals, _ = torch.sort(vals)
        mask = vals < density
        A = A.view(-1)
        A[mask] = 0.0
        A = A.view(cfg.N, cfg.N)
    elif cfg.pattern == "banded":
        i, j = torch.meshgrid(torch.arange(cfg.N), torch.arange(cfg.N))
        mask = torch.abs(i - j) <= int(cfg.N * (1 - density) / 2)
        A *= mask.float()
    elif cfg.pattern == "block_diagonal":
        block_size = int(cfg.N * (1 - density) ** 0.5)
        blocks = [torch.rand(block_size, block_size) for _ in range(cfg.N // block_size + 1)]
        A = torch.block_diag(*blocks)
        A = A[:cfg.N, :cfg.N]

    V = torch.randn(cfg.N, cfg.batch_size, dtype=torch.float32)
    return A, V

def normalize_for_hash(out: torch.Tensor) -> np.ndarray:
    arr = out.cpu().float().numpy()
    for j in range(arr.shape[1]):
        col = arr[:, j]
        m = col.mean()
        s = col.std() + 1e-8
        arr[:, j] = (col - m) / s
    return arr

def compute_hashes(out: torch.Tensor) -> Dict[str, str]:
    norm = normalize_for_hash(out)
    flat = norm.flatten()
    sha256 = hashlib.sha256(flat[:4_000_000].tobytes()).hexdigest()
    qhash = round(float(np.mean(norm)), 6)
    return {"sha256_norm": sha256, "qhash": str(qhash)}

def get_hardware_info() -> Dict[str, str]:
    gpu = torch.cuda.get_device_name(0) if torch.cuda.is_available() else "CPU only"
```

```
    ram_gb = round(os.sysconf('SC_PAGE_SIZE') * os.sysconf('SC_PHYS_PAGES') / (1024**3), 1) if
hasattr(os, 'sysconf') else "Unknown"
    return {
        "os": platform.system() + " " + platform.release(),
        "cpu": platform.processor() or platform.machine(),
        "gpu": gpu,
        "ram_gb": str(ram_gb),
        "torch_version": torch.__version__
    }

def run_verifier(cfg: TestConfig) -> Dict[str, Any]:
    A, V = generate_matrix(cfg)

    # Warmup
    for _ in range(cfg.warmup):
        _ = torch.mm(A, V)

    # Baseline timing
    start = time.perf_counter()
    for _ in range(cfg.iters):
        out = torch.mm(A, V)
    baseline_time_ms = (time.perf_counter() - start) / cfg.iters * 1000

    hashes = compute_hashes(out)
    hardware = get_hardware_info()

    result = {
        "config": {
            "N": cfg.N,
            "zeros_pct": cfg.zeros_pct,
            "pattern": cfg.pattern,
            "batch_size": cfg.batch_size,
            "iters": cfg.iters,
            "seed": cfg.seed,
            "baseline_method": "dense_torch_mm"   # change manually if you used vendor sparse
        },
        "hardware": hardware,
        "baseline": {
            "time_ms": round(baseline_time_ms, 4)
        },
        "hashes": hashes,
        "message": "Public verification kit — ZERO ROLV IP. Send this JSON for official ROLV comparison."
    }
    return result

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="ROLV Verification Kit")
    parser.add_argument("--N", type=int, default=20000, help="Matrix size")
    parser.add_argument("--zeros", type=float, default=0.70, help="Sparsity 0-1")
    parser.add_argument("--batch", type=int, default=5000, help="Batch size")
    parser.add_argument("--iters", type=int, default=1000, help="Iterations")
    parser.add_argument("--pattern", type=str, default="random")
    args = parser.parse_args()

    cfg = TestConfig(N=args.N, zeros_pct=args.zeros, batch_size=args.batch,
                     iters=args.iters, pattern=args.pattern)
```

```
result = run_verifier(cfg)

print(json.dumps(result, indent=2))
with open("rolv_baseline.json", "w") as f:
    json.dump(result, f, indent=2)
print("\n✅ Saved to rolv_baseline.json — please send this file to us!")
```

3. **Customize the test (optional)**
   Use command-line flags (no need to edit the file):

```
python rolv-verifier.py --N 16384 --zeros 0.92 --batch 2048 --iters 500 --pattern power_law
```

4. **Send us the rolv_baseline.json file (email to rolv@rolv.ai or upload link).**

We run the full proprietary ROLV harness on the exact same inputs and return a professional comparison report.

What you receive

A full professional report:

• Your baseline vs ROLV (speedup, energy savings, tokens/sec, FLOPs)
• Correctness verification (hashes match)
• All numbers on the exact same inputs and your exact hardware