Superpod Handoff Runbook

Single-Command Execution

Joe Corneli

February 16, 2026

1. Run Location

Run on the Superpod execution host after cloning `futon6`.

```
git clone <futon6-repo-url>
cd futon6
```

2. Single Command (required path)

```
bash scripts/handoff-superpod-all.sh
```

What it does (with step-by-step progress and hard fail gates):

- bootstrap inputs
- sanity tests
- smoke run + verification
- CPU baseline runs + verification
- required GPU backfill runs + verification
- packaging

3. Fast Verification Mode

```
bash scripts/handoff-superpod-all.sh --smoke-only
```

4. Deliverables

Expected outputs:

- `superpod-math-processed.tar.gz`
- `superpod-mo-processed.tar.gz`
- `superpod-math-processed-gpu.tar.gz`
- `superpod-mo-processed-gpu.tar.gz`

5. Return Payload

Send back:

- all 4 tarballs above
- short metric table from CPU and GPU `manifest.json` files:
- `entity_count`
- `stage5_stats.total_ner_hits`
- `stage7_stats.threads_processed`
- `stage7_stats.total_nodes`
- `stage7_stats.total_edges`
- `stage7_stats.n_categorical`
- `stage7_stats.n_port_matches`

6. Notes

- `-skip-bootstrap` and `-skip-tests` are available for reruns only.
- `scripts/handoff-superpod-all.sh` is the source of truth.
- Companion machine-readable note: `data/first-proof/superpod-handoff-rob.lit.md`

7. Mission Wiring Diagram (futon5 style)

Intent. Convert raw public math Q/A corpora into verified typed wiring artifacts that can be queried by downstream proof work.

Legend. `M*=control`, `D*=download/input`, `P*=process`, `V*=verify`, `O*=output`.

```
Mission Control (single command)
  M0: bash scripts/handoff-superpod-all.sh
      |
      +--> M1 bootstrap: scripts/handoff-superpod-bootstrap.sh
      |       |
      |       +--> D1 math.stackexchange.com (Posts.xml, Comments.xml)
      |       +--> D2 mathoverflow.net       (Posts.xml, Comments.xml)
      |       +--> D3 local refs (nlab-ct-reference.json, terms.tsv)
      |
      +--> M2 tests: pytest smoke + verifier checks
      |
      +--> M3 smoke gate (mini dataset, 4 threads)
      |       |
      |       +--> V1 verify required files + edges_checked > 0
      |
      +--> M4 CPU baseline run (math.SE + MO; stages 1/5/7)
      |       |
      |       +--> P1 parse XML posts/comments
      |       +--> P2 NER + scope extraction
      |       +--> P3 CT-backed thread wiring assembly
      |       +--> V2 manifest sanity + ct-verifier
      |       +--> O1 math-processed/
      |       +--> O2 mo-processed/
      |
      +--> M5 required GPU backfill (math.SE + MO; full stages 1..7)
      |       |
      |       +--> P4 embeddings (GPU)
      |       +--> P5 LLM pattern tagging (GPU)
      |       +--> P6 clustering + reverse morphogenesis
      |       +--> V3 ct-verifier refresh
      |       +--> O3 math-processed-gpu/
      |       +--> O4 mo-processed-gpu/
      |
      +--> M6 packaging
              |
              +--> O5 superpod-math-processed.tar.gz
              +--> O6 superpod-mo-processed.tar.gz
              +--> O7 superpod-math-processed-gpu.tar.gz
              +--> O8 superpod-mo-processed-gpu.tar.gz
```

Invariants enforced by the orchestrator.

- Required artifacts exist: manifest, CT wiring output, and CT verifier output.
- `stage7_stats.ct_backed=true`, `stage7_stats.threads_processed > 0`, `edges_checked > 0`; otherwise fail hard.

8. Why this work is interesting and valuable

This run is not just data collection. It produces a reusable evidence layer for math reasoning work: each thread becomes a typed wiring object with explicit nodes, edges, and port matches, not just text.

That is valuable for two reasons. First, retrieval quality improves: we can ask for threads that match an input/output proof shape, not only threads that share keywords. Second, verification quality improves: `ct-verifier` checks the resulting structure and blocks empty artifacts.

The CPU and GPU outputs are complementary, not interchangeable. CPU gives a deterministic baseline and immediate wiring products. GPU backfill adds richer semantic signals (embeddings and LLM-derived structure) over the same corpus. Keeping both lets us compare quality and track where extra compute changes results.

At project level, this turns a one-off run into infrastructure. The same pipeline can be rerun, audited, and diffed over time, so Rob can evaluate whether later changes improve signal, regress quality, or break invariants.