Superpod Run

What It Is and What It Produces

A Handoff Note

Joe Corneli | Hyperreal Enterprises | February 15, 2026

Context

Rob — this note explains the "superpod run," a batch processing job that takes the full StackExchange mathematics data dumps and transforms them into structured knowledge artifacts. The run is the next concrete step in the futon6 project, and its output feeds directly into preparations for First Proof Batch 2 (expected ~1 month from now).

The short version: we're taking 567K math.stackexchange threads and 100K MathOverflow threads, running them through a 7-stage pipeline, and producing typed wiring diagrams for every thread — machine-readable representations of how the arguments in each thread fit together, what mathematical structures they reference, and where the logical connections are.

What Goes In

StackExchange publishes complete data dumps of every Q&A site. For mathematics we use two:

- **math.stackexchange.com —** 567K question threads. Undergraduate through graduate level. High volume, broad coverage.
- **mathoverflow.net —** 100K question threads. Research level. Narrower but deeper.

Each thread has a question, zero or more answers, and comments. The raw data is XML. Total: roughly 20 GB, compressing to about 5 GB.

What the Pipeline Does

The superpod job has seven stages. Not all require a GPU — the first, fifth, and seventh run on a laptop.

#	Stage	What it does	HW
1	**Parse**	Stream XML into structured QA pairs	CPU
2	**Embed**	Dense 768-dim vector embedding of every post (BGE-large)	GPU
3	**Tag**	LLM pattern tagging: 25 argument patterns (Llama-3-8B)	GPU
4	**Cluster**	Group threads by topic using HDBSCAN on embeddings	CPU
5	**NER + Scopes**	Named-entity recognition (19K terms) + discourse detection	CPU
6	**Rev. morph.**	LLM reconstruction of logical skeleton per QA pair	GPU
7	**Thread wiring**	CT-backed wiring diagram: IATC edges, ports, categorical	CPU

Three run modes: Full (GPU, all stages), Moist (CPU stages + prompt files for cloud LLM handoff), CPU-only (Stages 1, 4, 5, 7). For the immediate run we'll use CPU-only or moist mode. The critical output is Stage 7.

What Comes Out

For every thread, Stage 7 produces a JSON wiring diagram with three levels:

1. **Thread level (discourse).** Each post becomes a node. Edges are typed by illocutionary act — assert, challenge, clarify, reference, retract, reform, etc. These capture the argumentative moves in the thread.
2. **Categorical level.** Each node is checked against a reference dictionary of 8 category-theory pattern types (adjunction, equivalence, fibration, etc.), extracted from 20K nLab wiki pages. IDF weighting avoids false positives.
3. **Port level.** Mathematical content is decomposed into input ports (assumptions, let-bindings) and output ports (conclusions, constraints). Edges carry port matches showing which outputs connect to which inputs, with scores boosted by CT reference weights.

So where a raw SE thread looks like "question → answer → comment," the wiring diagram looks more like: "question introduces terms X, Y via let-bindings; answer asserts conclusion Z via Cauchy-Schwarz referencing X; comment challenges the bound on Y with an adversative discourse marker."

Scale

	math.SE	MathOverflow
Threads	~567K	~100K
Nodes (posts)	~3M	~500K
Edges (typed)	~2.5M	~400K
Output size (est.)	~15 GB	~3 GB

The 200-thread pilot we ran this week produced 1,641 nodes, 1,441 edges, and 319 categorical detections with sharp differentiation: category-theory threads averaged 3 categorical patterns per thread vs. 0.2 for mathematical-physics threads. The signal is real.

How This Connects to First Proof 2

We did a proof sprint on the First Proof benchmark two weeks ago (10 research-level math problems, 55 hours, 4/10 correct). A detailed retrospective identified five things we'd do differently. The superpod run addresses three of them directly:

1. **Better literature mining.** During Sprint 1, keyword-based searches missed critical techniques (e.g., hyperbolic Hessian contraction bounds for Problem 4). The superpod output provides a structured index of how 667K threads discuss mathematical concepts — not just keywords, but typed connections between assumptions and conclusions. When we need a bridge between proof steps, we can query: "what threads have an output port matching this input?"
2. **Automated proof verification.** We built a verifier that checks each edge in a proof wiring diagram for structural consistency (port types, discourse markers, categorical patterns). During Sprint 1 these checks were manual. Now they're automated, and the superpod gives the verifier a large corpus to calibrate against.
3. **Domain-depth pre-loading.** Sprint 1's two wrong answers were both on problems requiring deep specialized knowledge. The superpod output lets us pre-identify which subfields have rich coverage and which are

sparse, so we can front-load targeted mining for the harder problems.

The other two lessons (mandatory falsification protocol, better calibration tracking) are process improvements that don't depend on the corpus but benefit from it — a verification score that plateaus is a concrete trigger for switching from constructive to falsification mode.

Timeline

- **This week:** Launch the CPU-only run on math.SE + MathOverflow (~48 hours). This produces parsed threads, NER + scopes, and CT-backed wiring diagrams.
- **Next 2 weeks:** Backfill GPU stages if cloud compute is available. Build a "proof assistant" wrapper integrating the verifier with the sprint workflow.
- **Week 4:** First Proof Batch 2, if announced. Otherwise, dry-run the protocol on a practice problem (e.g., re-attempt Problem 4 targeting the all-n bridge we missed the first time).

What I'd Like From You

Mainly: a sanity check. Does this pipeline make sense as a way to build structured mathematical knowledge at scale? The bet is that typed wiring diagrams (not just embeddings, not just keyword search) are the right intermediate representation — rich enough to support automated verification, structured enough to enable port-level queries, but not so formal that we need full theorem-proving infrastructure.

If you want to poke at any of the artifacts, the repo is futon6 on GitHub (private, happy to add you). Key files:

```
scripts/superpod-job.py          # the pipeline (1,700 lines)
scripts/assemble-wiring.py       # wiring assembly (750 lines)
scripts/ct-verifier.py           # proof verifier (570 lines)
data/nlab-ct-reference.json      # CT reference (20K pages)
```