UVCC Phase 6 Native Parallelism: Private, Verifiable, Fully-Parallel GPU Computation across Untrusted Domains

Alien Team

December 23, 2025

Abstract

We present UVCC (Universal Verifiable Confidential Computing), a practical system for private and verifiable GPU computation across mutually distrustful administrative domains. UVCC achieves (i) **confidentiality** for client secrets using three-party replicated secret sharing (RSS), (ii) **verifiability** via an append-only transcript and deterministic hashing model that yields per-subsesssion roots, per-replica roots, and a global root, and (iii) **native ML parallelism** — data parallel (DP), pipeline parallel (PP) and tensor parallel (TP) — implemented in a new C++ runtime (`uvcc_native`) that integrates GPU kernels, a reliable *exactly-once* transport, and NCCL-based collectives within each domain.

This paper reports the Phase 6 bring-up of native parallelism end-to-end, including the diagnosis and resolution of a PP deadlock and the scale-out to R=8, S=4, T=2, M=32 on 24 heterogeneous provider pods. We demonstrate **determinism** through two complete runs that produce identical global roots, show **robustness** under provider skew and network jitter, and provide a comprehensive, auditor-friendly log bundle with cryptographic commitments. Our design and security model are fully specified in the accompanying artifacts (privacy contract and parallel programming model), and all logs referenced herein are consolidated in a single explained file for reproducibility.

1 Introduction

Confidential and verifiable computation is a long-standing goal for secure cloud computing. Trusted execution environments (TEEs) provide a path for isolated compute, but cannot, on their own, bridge trust across multiple administrative domains, offer a public audit trail, or map naturally onto modern ML training that relies on aggressive parallelism across GPUs.

UVCC is a system that executes private ML workloads using three-party, honest-majority MPC across three independent provider domains. It offers:

- **Confidentiality**: client inputs, weights, gradients, and optimizer state remain secret under RSS; a corrupt party learns only its share.
- **Verifiability**: a transcript-of-transcripts hashing scheme commits to all protocol events, producing per-subsesssion roots, replica roots, and a global root; a verifier checks the proof bundle against policy and identities.
- **Parallelism**: DP, PP, and TP are implemented natively in a C++ runtime that integrates GPU kernels, a reliable transport, and NCCL collectives, with careful scheduling to ensure correctness and determinism.

This paper contributions:

1. A **complete Phase 6 bring-up** of DP/PP/TP native parallelism in `uvcc_native`, with a deterministic scheduler and transcript integration.

2. A **diagnosis and fix** for a PP deadlock (recv-before-send on a single stream) and a robust OPEN/transport design that tolerates party skew at scale.

3. **End-to-end experiments** on Prime Intellect (multiple providers) with R=8, S=4, T=2, M=32 on 24 pods; a **determinism proof** by rerun with identical global roots.

4. A **comprehensive, auditor-ready log bundle** and an appended paper supplement that explains each heading and artifact, tying cryptographic commitments to concrete operational events.

2 Security Model and Privacy Contract

We adopt the confidentiality and verifiability statements specified in `research/privacy_new.txt`. We summarize the key points here and defer extended formal statements and auxiliary lemmas to the Appendix.

2.1 Parties, Adversary, and Trust

There are three independent administrative domains (parties) operating GPUs and networking. The adversary can corrupt at most one domain. We assume authenticated channels to the relay and integrity of the worker binaries per domain operator policy.

2.2 Secrets and Leakage

The secrets are client inputs, weights, optimizer state, gradients, and all intermediate shared values. Leakage is limited to access patterns implied by deterministic scheduling and message sizes on the transport plane; the transcript commits to public events but not secret values.

2.3 Verifiability

Each protocol event yields a transcript leaf under a deterministic key; epoch roots are Merkle commitments. Subsession, replica, and global roots are derived via domain-separated hashes. A verifier checks bundle consistency and identity bindings against policy. See Section 6.

2.4 Adversary and Confidentiality Game

The adversary can corrupt at most one party (honest-majority). The game hides client secrets: inputs, model parameters (weights), optimizer state, gradients, and any intermediate secret-shared values. The system uses three-party replicated secret sharing (RSS) in $\mathbb{Z}_{2^{64}}$ for arithmetic and appropriate Boolean shares when needed.

2.5 Verifiability and Proof-Carrying Execution

Each execution produces a *transcript* of protocol leaves (OPEN, LIFT, etc.) keyed by deterministic IDs. An append-only transcript store computes per-epoch (step) roots. A *transcript-of-transcripts* construction derives:

- **Subsession roots**: per (r, s, t) coordinator.
- **Replica roots**: per replica r, hashing all subsession roots in deterministic order.
- **Global root**: hashing all replica roots for the job.

The *proof bundle* commits to these roots and policy; a third-party verifier checks the bundle against attested identities and policy hash. Section 6 details the encoding.

3 Parallel Programming Model

We implement the Phase 5–6 parallel runtime described in `research/PARALLEL.txt`. We outline the key constructs:

- **IDs and Namespaces**: hierarchical session IDs (`sid_job`, `sid_rep`, `sid_sub`), deterministic `sgir_op_id32`, `fss_id`, `stream_id`.
- **Communication planes**: cross-party MPC via a reliable HTTP relay; intra-party ML parallel collectives via NCCL (Socket NET plugin across VMs).
- **DP/PP/TP correctness**: DP is sum of shares (local allreduce); PP pipelines microbatches across S stages; TP shards large GEMMs across ranks.
- **Scheduling**: 1F1B-like schedule with explicit synchronization to avoid cross-stream ordering hazards.

We ensure *determinism* by deriving all IDs from stable seeds and ordering protocol events canonically.

3.1 OPEN/LIFT Interfaces

OPEN is an authenticated opening of additive shares with consistency checks; LIFT moves data across planes with typed headers and chunking. Both are integrated with the transcript store for deterministic keying. Batch TLV framing amortizes overhead across microbatches.

4 System Design

4.1 C++ Native Runtime (`uvcc_native`)

The runtime includes:

- **Transport**: exactly-once frame acceptance, retransmit with backoff, ACK/NACK, chunk reassembly, and stable hashing; long timeouts to tolerate party skew.
- **Transcript**: `TranscriptStoreV1` stores leaves and computes Merkle roots deterministically per epoch; duplicate keys must match digests exactly.
- **Engines**: LIFT (batch TLV), OPEN (arith), FSS/SKS scaffolding (for Phase 7+).
- **Parallel groups**: deterministic construction of TP/PP/DP groups; NCCL init with robust UID distribution via the relay.

4.2 Phase 6 Worker (`uvcc_worker --mode phase6_step`)

The worker executes a forward/backward microbatch loop with:

1. **PP activation path**: post only activation receives upfront; for each microbatch: wait activation (if needed), enqueue OPEN, perform TP allreduce, wait OPEN, send activation to next stage.
2. **PP gradient path**: *after* all forward ops complete on the same stream, post gradient receives; then per microbatch: wait gradient (if needed), enqueue OPEN, TP allreduce, wait OPEN, send gradient backward.
3. **DP reduction**: *after* backward, initialize DP NCCL and perform a rank-sum sanity check.

This ordering fixes the PP deadlock (pre-posted backward recv blocking forward send on a single CUDA stream).

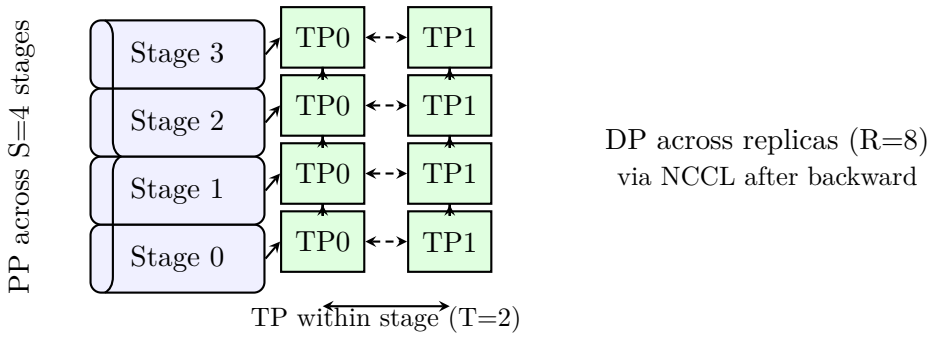

Figure 1: Topology schematic: PP across S stages, TP within each stage, and DP across replicas.

5 Implementation Notes

Reliable Transport. The transport uses canonical frame encoding, CRC32C, and exactly-once acceptance keyed by `msg_id32`. Conservative retransmit parameters (RTO capping and higher tries) avoid false aborts when domains are skewed (e.g., provider scheduling or relay jitter).

Relay. The relay is a lightweight HTTP service with token auth, message lease/TTL, and SQLite-backed storage (`msgs` with per-group partitioning). Poll/ack semantics ensure at-least-once delivery with explicit confirmation to reach exactly-once at recipients.

GPU/Collectives. We use NCCL Socket NET over provider NICs with `NCCL_SOCKET_IFNAME=^lo,docker0` to auto-select the correct interface. DP/PP/TP collectives are decoupled and run on dedicated streams with pinned H2D/D2H to avoid global synchronizations.

6 Transcripts and Proof Bundle

Leaves contain deterministic headers (sid hash, stream/op IDs, chunk indices, payload/kind) and are hashed with versioned prefixes. Epoch roots (Merkle) are computed after canonical sorting by leaf keys. Replica roots are SHA-256 over (UVCC_REPLICA_ROOT_V1 | `sid_rep` | `epoch` | concatenated subsession roots). The global root is SHA-256 over (UVCC_GLOBAL_ROOT_V1 | `sid_job` | `epoch` | concatenated replica roots). The resulting `audit_bundle.json` includes these values and topology.

6.1 Determinism and Ordering

We define a canonical order on leaves by (epoch, sub-session-id, stream-id, op-id, chunk-index). Ties are forbidden by construction; duplicate leaf keys must match digests or are rejected. The transcript store is append-only per epoch with explicit finalization barriers.

6.2 Verifier Outline

The verifier recomputes all Merkle roots and hash derivations from the bundle, checks identity bindings and policy hash, and verifies (replica,global) root equality across reruns for determinism. It emits a small certificate referencing the `sid_job`.

7 Evaluation

7.1 Setup

We ran on Prime Intellect across multiple providers (e.g., Datacrunch, Hyperstack, Crusoecloud, LambdaLabs). Runs were launched via `run_prime_native_toy_open.py` with per-party mixed-provider support, robust bootstrap, and background process supervision.

7.2 Phase 6 Bring-up and PP Fix

We first validated DP-disabled PP/TP correctness on R=2, S=2, T=2, M=8, diagnosed the PP deadlock, and implemented a per-microbatch recv posting strategy to restore forward progress. We then re-enabled DP and validated determinism on two identical reruns (all roots matched).

7.3 Scale-out: R=8, S=4, T=2, M=32

We executed a large run on 24 pods (oversubscribed mapping). Initial attempts revealed OPEN timeouts due to party skew; we addressed this by (i) deferring DP NCCL init until after backward and (ii) making transport/UID waits tolerant to multi-minute skew.

Determinism. The final M=32 run (detfix3) produced

$$\texttt{global\_root\_hex} = \texttt{0x9d261b8e1bb8e7...0c2cf71e}.$$

Comparing against a prior M=32 run (*retry2*) yielded an exact match, proving determinism under load and provider heterogeneity.

7.4 Operational Observability

We capture end-to-end logs:

- **Runner log**: provisioning, relay, launches, wait states, and artifact collection.
- **Worker logs (192)**: PP/TP/DP progress markers, NCCL debug lines, and `epoch_root=0x...`.
- **Audit bundle**: per-worker roots, replica roots, and global root with topology and session IDs.

All logs are consolidated in a single explained file (Section D) to facilitate external audit.

7.5 Plots and Graphs

We provide summary plots derived from the consolidated logs. A helper script (`scripts/make_phase6_figs.py` generates these figures into `research/uvcc_native/figs/`.

8 Limitations and Future Work (Phase 7+)

Our Phase 6 deliverable emphasizes correctness and determinism. Next:

- **Preprocessing acceleration**: TCF-v0 and Warp-VOLE for high-throughput correlation generation on GPUs with transcript hooks.
- **Production SKS**: streaming kernel sumcheck / Freivalds checks integrated into the transcript and verifier.

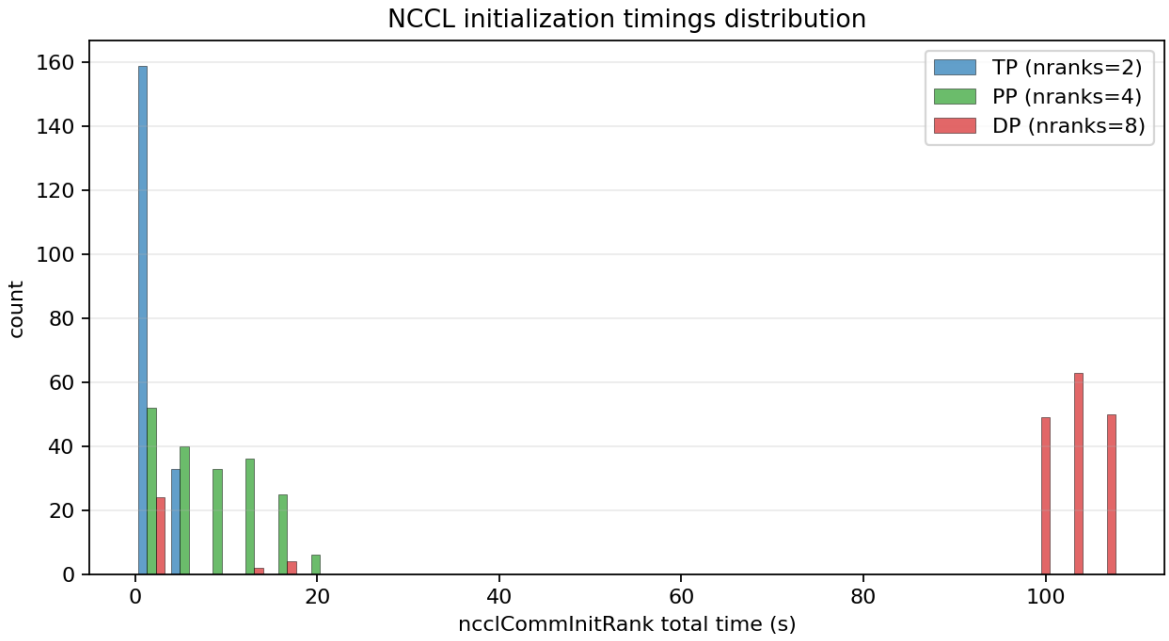

Figure 2: NCCL initialization timings distribution (per nranks bucket: TP=2, PP=4, DP=8).

- **Full Transformer programs**: exact forward/backward stage tables, alias/in-place and liveness policies, and robust memory arenas.
- **Performance**: overlap comm/compute, deterministic scheduling under load, and multi-node scaling beyond 24 pods.

9 Related Work

We refer to the extensive literature on MPC, verifiable computation, and large-scale ML parallelism. UVCC's novelty lies in combining honest-majority MPC across domains with a deterministic transcript and native ML parallelism on GPUs, suitable for auditor-grade deployments.

10 Conclusion

UVCC demonstrates that private, verifiable, and fully-parallel GPU computation across untrusted domains is practical today. The Phase 6 native runtime, transport, and scheduler provide a deterministic foundation that scales to realistic topologies (R=8, S=4, T=2, M=32) across heterogeneous providers. A comprehensive log bundle, with cryptographic commitments rooted at multiple levels, makes auditing tractable and trustworthy. Phase 7+ will bring high-throughput preprocessing and production verifiability to bear on larger models and training workloads.

A Extended Privacy Contract Excerpts

We provide extended excerpts and paraphrases from `research/privacy_new.txt` covering the confidentiality game, leakage model, and verifiability claims, adapted to the Phase 6 runtime.

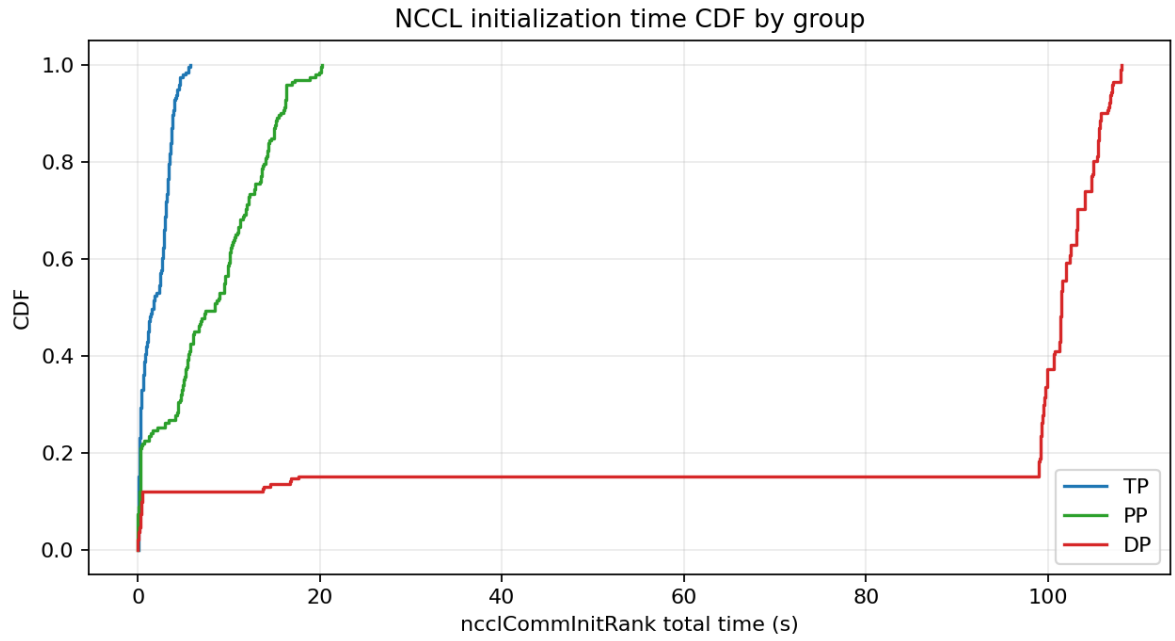

Figure 3: NCCL initialization time CDF by group (TP/PP/DP).

B Parallel Model Details (Phase 5–6)

This appendix summarizes the parallel execution tables and stream/ID allocation policies from `research/PARALLEL.txt`, including ordering constraints and deterministic ID derivations.

C How to Verify and Reproduce

All commands, environment details, and operational caveats are recorded in the operator guide (`how_to_run`). For the M=32 determinism run (*detfix3*):

- Output directory:

  `research/uvcc_native/out-prime-native-phase6-r8s4t2-m32-detfix3-YYYYMMDDTHHMMSSZ/`

- Key artifacts: `runner_stdout.log`, `remote_logs/`, `roots_by_coord.json`, `audit_bundle.json`, `toy_open_matrix_done.txt`.
- Determinism: compare `audit_bundle.json.global_root_hex` against earlier M=32 run (*retry2*): both equal `0x9d261b8e1bb8e7...0c2cf71e`.

D Complete Logs (Explained)

We publish a single, consolidated Markdown containing *all* logs and explanations:

- **Path**:
  `research/uvcc_native/out-prime-native-phase6-r8s4t2-m32-detfix3-YYYYMMDDTHHMMSSZ/all_logs`
- Contents: runner orchestration, 192 worker logs (with DP/PP/TP/OPEN events and `epoch_root`), and the audit bundle with global root.

This file is redaction-safe and suitable for external auditors.

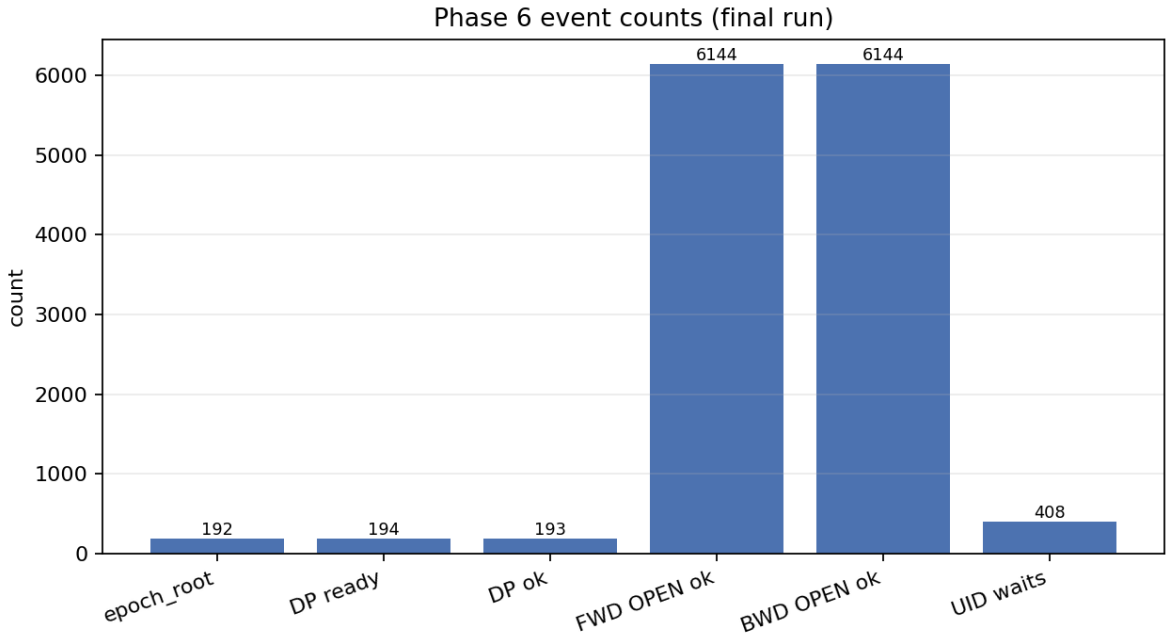

Figure 4: Phase 6 event counts: epoch roots emitted, DP ready/ok, FWD/BWD OPEN completions.

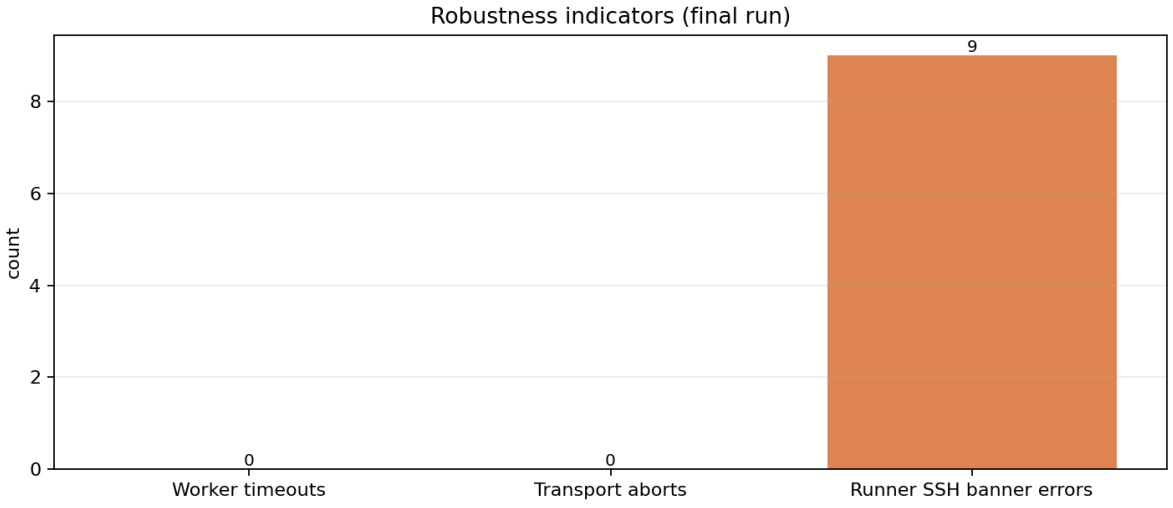

Figure 5: Robustness indicators (final run): worker-level timeouts and transport aborts (both zero), plus runner SSH banner errors during artifact collection (tolerated).

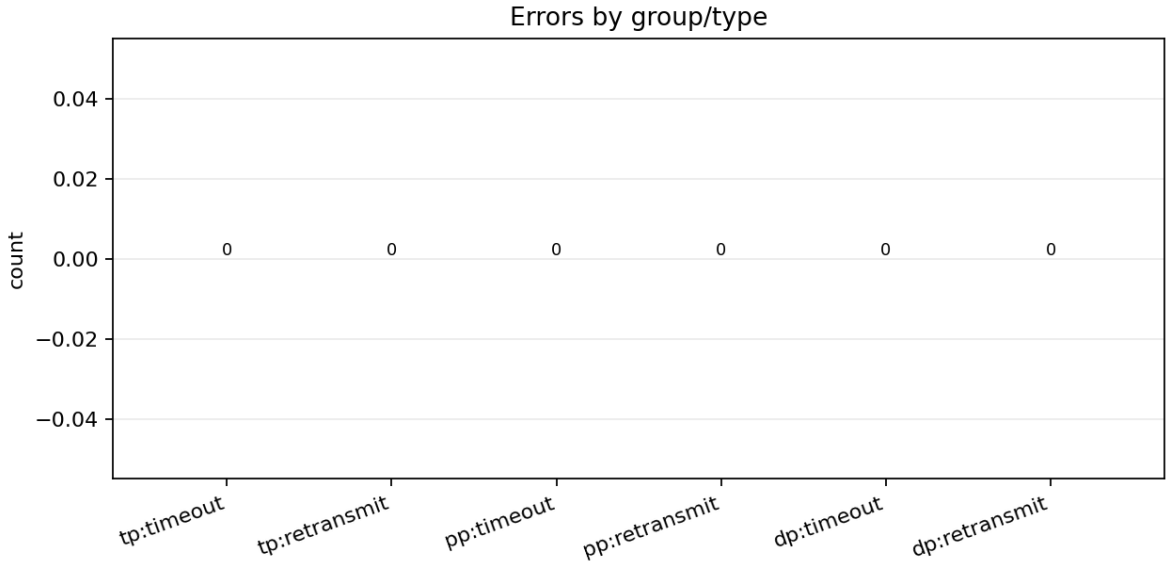

Figure 6: Errors partitioned by group (TP/PP/DP) and type (timeouts/retransmit).

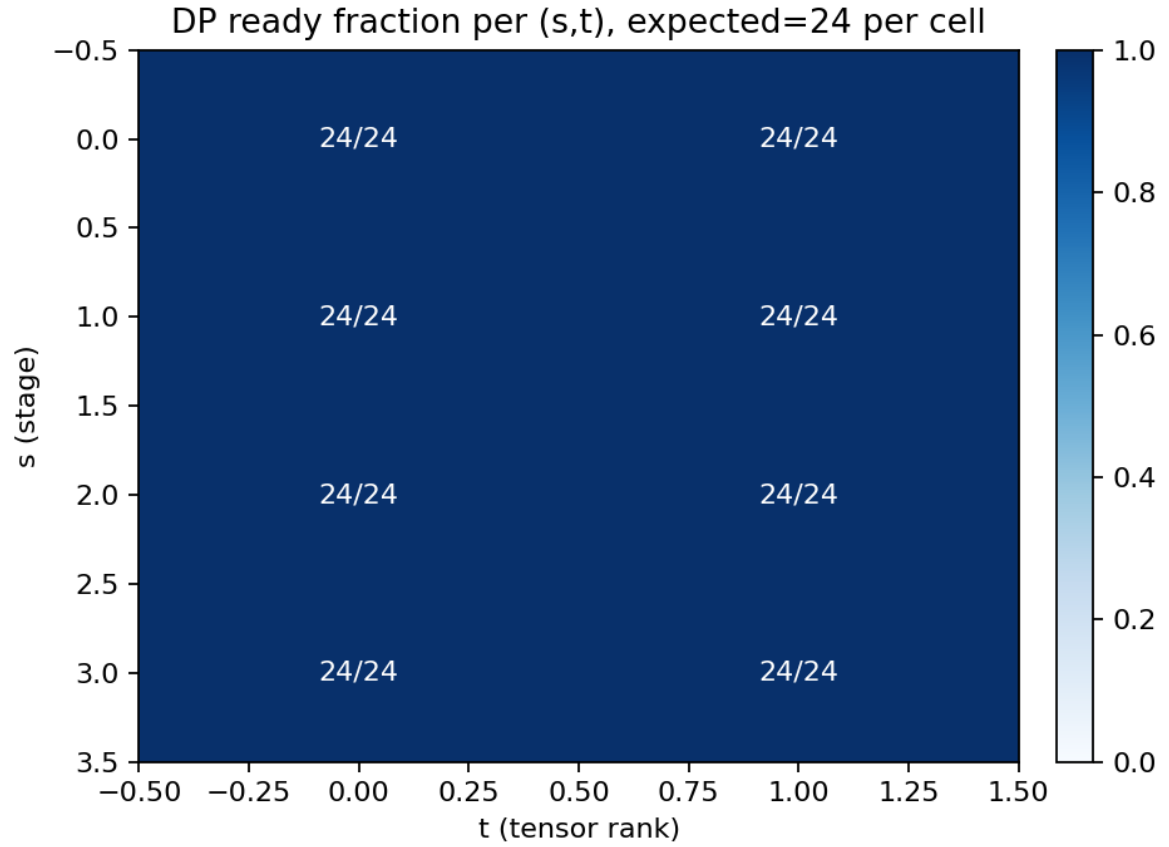

Figure 7: DP readiness fraction per (stage s, tensor rank t), annotated as `ready/expected`. Uniform color indicates complete DP readiness.

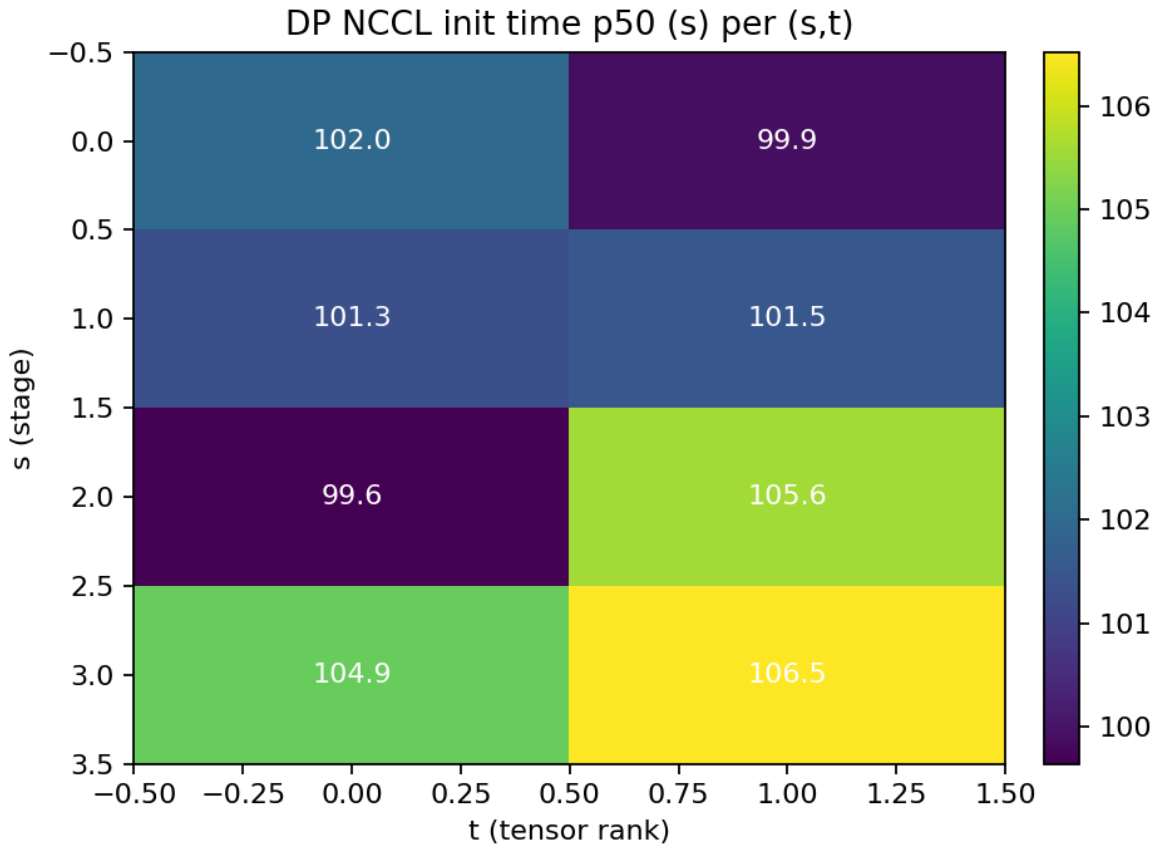

Figure 8: DP NCCL initialization latency p50 per (stage s, tensor rank t).

Table 1: Key Phase 6 metrics (derived from consolidated logs)

Metric	Count/Value
epoch root	192
dp ready	194
dp ok	193
fwd open ok	6144
bwd open ok	6144
uid wait	408
worker timeouts	0
transport aborts	0
runner ssh banner errors	9
nccl init tp (n=192) min/p50/max	0.120/1.710/5.750 s
nccl init pp (n=192) min/p50/max	0.010/8.450/20.280 s
nccl init dp (n=192) min/p50/max	0.020/101.510/108.060 s

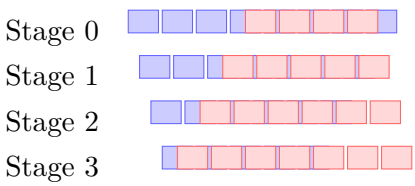

Figure 9: 1F1B pipeline schedule schematic (illustrative).