System Correctness, Oracles, and Replay

Deep Dive into DeepSeek — Chapter 34

Yin Yang

Foundation Books Project

The one rule of this chapter

An optimization may change *representation*, *placement*, or *timing*.
The verdict still depends on retained identity metadata
reaching a named oracle.

- Optimized systems can fail without crashing: shape is preserved, the text stays fluent, and the bytes attach to the wrong token, position, scale, owner, or version.
- Correctness and throughput are orthogonal. A path can preserve every property and still be too slow, or violate them at an attractive speed.
- So a test needs three things a benchmark does not: an independent reference, identity metadata that survives the transformation, and a replay record that reconstructs the failing case.

The one rule of this chapter

An optimization may change *representation*, *placement*, or *timing*.
The verdict still depends on retained identity metadata
reaching a named oracle.

- Optimized systems can fail without crashing: shape is preserved, the text stays fluent, and the bytes attach to the wrong token, position, scale, owner, or version.
- Correctness and throughput are orthogonal. A path can preserve every property and still be too slow, or violate them at an attractive speed.
- So a test needs three things a benchmark does not: an independent reference, identity metadata that survives the transformation, and a replay record that reconstructs the failing case.

Goal: build one four-token MoE oracle by hand, then carry the same identity–oracle–replay recipe to kernels, caches, distributed protocols, and repository interfaces — every verdict tied to the input domain, revision, and environment actually tested.

Roadmap

1. Correctness surfaces

Roadmap

1. Correctness surfaces
2. The four-token oracle

Roadmap

1. Correctness surfaces
2. The four-token oracle
3. Oracles and independent references

Roadmap

1. Correctness surfaces
2. The four-token oracle
3. Oracles and independent references
4. Distributed replay

Roadmap

1. Correctness surfaces
2. The four-token oracle
3. Oracles and independent references
4. Distributed replay
5. Cross-surface handoffs

Roadmap

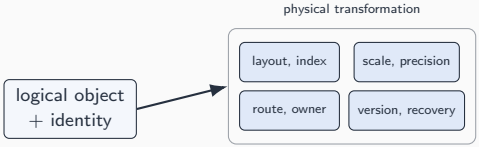
1. Correctness surfaces
2. The four-token oracle
3. Oracles and independent references
4. Distributed replay
5. Cross-surface handoffs
6. Repository interfaces and protocol scope

Correctness surfaces

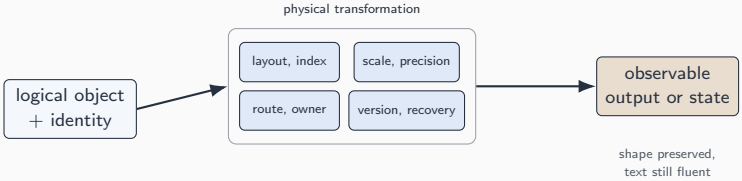
One logical object, many physical forms

logical object
+ identity

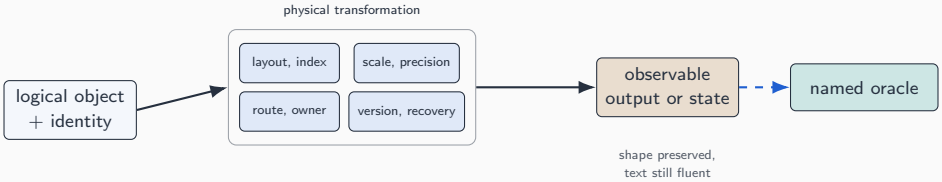
One logical object, many physical forms



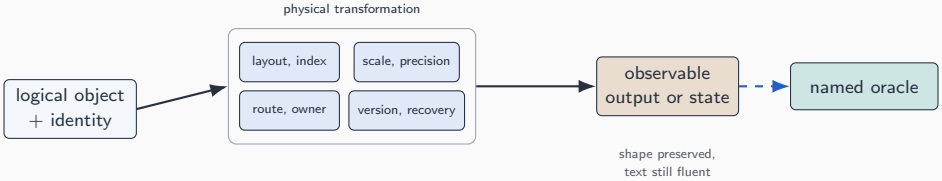
One logical object, many physical forms



One logical object, many physical forms

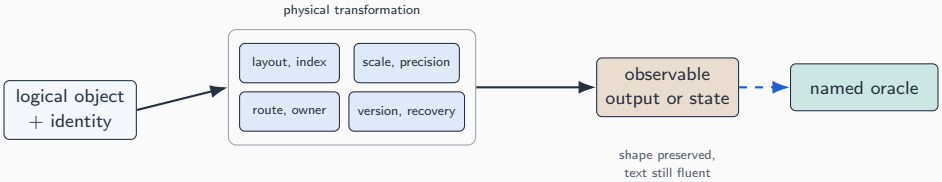


One logical object, many physical forms



Token rows become cache entries, routed expert work, kernel operands, communication buffers, and restored outputs.

One logical object, many physical forms



Token rows become cache entries, routed expert work, kernel operands, communication buffers, and restored outputs.

Correctness testing asks one question: did the intended object survive those transformations?

The unit of testing is a surface, not a function

Definition (Correctness surface)
A correctness surface names a component and valid input domain, the output or state to observe, the metadata needed to interpret it, and the oracle that decides whether the behavior is acceptable.

Four obligations the book states in exactly this form:

- A sparse-attention kernel must gather the intended cache positions.

The unit of testing is a surface, not a function

Definition (Correctness surface)
A correctness surface names a component and valid input domain, the output or state to observe, the metadata needed to interpret it, and the oracle that decides whether the behavior is acceptable.

Four obligations the book states in exactly this form:

- A sparse-attention kernel must gather the intended cache positions.
- Dispatch and combine must return expert results to their origin rows.

The unit of testing is a surface, not a function

Definition (Correctness surface)
A correctness surface names a component and valid input domain, the output or state to observe, the metadata needed to interpret it, and the oracle that decides whether the behavior is acceptable.

Four obligations the book states in exactly this form:

- A sparse-attention kernel must gather the intended cache positions.
- Dispatch and combine must return expert results to their origin rows.
- A low-precision kernel must interpret the *matching* scales.

The unit of testing is a surface, not a function

Definition (Correctness surface)
A correctness surface names a component and valid input domain, the output or state to observe, the metadata needed to interpret it, and the oracle that decides whether the behavior is acceptable.

Four obligations the book states in exactly this form:

- A sparse-attention kernel must gather the intended cache positions.
- Dispatch and combine must return expert results to their origin rows.
- A low-precision kernel must interpret the *matching* scales.
- A storage retry must not overwrite newer content.

The unit of testing is a surface, not a function

Definition (Correctness surface)
A correctness surface names a component and valid input domain, the output or state to observe, the metadata needed to interpret it, and the oracle that decides whether the behavior is acceptable.

Four obligations the book states in exactly this form:

- A sparse-attention kernel must gather the intended cache positions.
- Dispatch and combine must return expert results to their origin rows.
- A low-precision kernel must interpret the *matching* scales.
- A storage retry must not overwrite newer content.

Each is an identity claim before it is a numerical claim.

The four-token oracle

A fixture whose answer exists without the system

Origin token g	Route 0 (e, w)	Route 1 (e, w)
0	$(0, 0.75)$	$(2, 0.25)$

Four rows $x_0 = (1, 0)$, $x_1 = (0, 1)$, $x_2 = (1, 1)$, $x_3 = (2, -1)$, top-2 routing, and the deliberately simple expert map $F_e(x) = x + (e, -e)$.

$$z_g = \sum_{j=0}^{k_g - 1} w_{g,j}\, F_{e_{g,j}}(x_g), \qquad z_0 = 0.75\, F_0(1, 0) + 0.25\, F_2(1, 0) = (1.5, -0.5).$$

A fixture whose answer exists without the system

Origin token g	Route 0 (e, w)	Route 1 (e, w)
0	$(0, 0.75)$	$(2, 0.25)$
1	$(1, 0.60)$	$(2, 0.40)$

Four rows $x_0 = (1, 0)$, $x_1 = (0, 1)$, $x_2 = (1, 1)$, $x_3 = (2, -1)$, top-2 routing, and the deliberately simple expert map $F_e(x) = x + (e, -e)$.

$$z_g = \sum_{j=0}^{k_g - 1} w_{g,j}\, F_{e_{g,j}}(x_g), \qquad z_0 = 0.75\, F_0(1, 0) + 0.25\, F_2(1, 0) = (1.5, -0.5).$$

A fixture whose answer exists without the system

Origin token g	Route 0 (e, w)	Route 1 (e, w)
0	$(0, 0.75)$	$(2, 0.25)$
1	$(1, 0.60)$	$(2, 0.40)$
2	$(0, 0.50)$	$(1, 0.50)$

Four rows $x_0 = (1, 0)$, $x_1 = (0, 1)$, $x_2 = (1, 1)$, $x_3 = (2, -1)$, top-2 routing, and the deliberately simple expert map $F_e(x) = x + (e, -e)$.

$$z_g = \sum_{j=0}^{k_g - 1} w_{g,j}\, F_{e_{g,j}}(x_g), \qquad z_0 = 0.75\, F_0(1, 0) + 0.25\, F_2(1, 0) = (1.5, -0.5).$$

A fixture whose answer exists without the system

Origin token g	Route 0 (e, w)	Route 1 (e, w)
0	$(0, 0.75)$	$(2, 0.25)$
1	$(1, 0.60)$	$(2, 0.40)$
2	$(0, 0.50)$	$(1, 0.50)$
3	$(2, 0.70)$	$(1, 0.30)$

Four rows $x_0 = (1, 0)$, $x_1 = (0, 1)$, $x_2 = (1, 1)$, $x_3 = (2, -1)$, top-2 routing, and the deliberately simple expert map $F_e(x) = x + (e, -e)$.

$$z_g = \sum_{j=0}^{k_g - 1} w_{g,j}\, F_{e_{g,j}}(x_g), \qquad z_0 = 0.75\, F_0(1, 0) + 0.25\, F_2(1, 0) = (1.5, -0.5).$$

A fixture whose answer exists without the system

Origin token g	Route 0 (e, w)	Route 1 (e, w)
0	$(0, 0.75)$	$(2, 0.25)$
1	$(1, 0.60)$	$(2, 0.40)$
2	$(0, 0.50)$	$(1, 0.50)$
3	$(2, 0.70)$	$(1, 0.30)$

Four rows $x_0 = (1, 0)$, $x_1 = (0, 1)$, $x_2 = (1, 1)$, $x_3 = (2, -1)$, top-2 routing, and the deliberately simple expert map $F_e(x) = x + (e, -e)$.

$$z_g = \sum_{j=0}^{k_g - 1} w_{g,j}\, F_{e_{g,j}}(x_g), \qquad z_0 = 0.75\, F_0(1, 0) + 0.25\, F_2(1, 0) = (1.5, -0.5).$$

Reference tensor: $Z_{\text{ref}} = \big((1.5, -0.5), (1.4, -0.4), (1.5, 0.5), (3.7, -2.7)\big)$ — in origin order.

[DERIVED: scaled stage view for hand calculation, not a released-model trace]

Two invariants do the work

Invariant (Route-count conservation)
A slot-expanded representation contains $\sum_g k_g$ logical route slots. A compact representation may send fewer physical token rows, but it must retain metadata that recovers the same route slots.

Invariant (Origin-order preservation)
Restored row g is the weighted result for origin token g, independent of the intermediate row number created by grouping or communication.

Two invariants do the work

Invariant (Route-count conservation)
A slot-expanded representation contains $\sum_g k_g$ logical route slots. A compact representation may send fewer physical token rows, but it must retain metadata that recovers the same route slots.

Invariant (Origin-order preservation)
Restored row g is the weighted result for origin token g, independent of the intermediate row number created by grouping or communication.

Count conservation catches missing or duplicated routes; origin order catches correct values returned to the wrong token.

Shape accepts all three outcomes

Case	Shape	Numerical symptom	Invariant result	Verdict
Reference	4×2	all four rows equal Z_{ref}	both invariants hold	Accept

Shape accepts all three outcomes

Case	Shape	Numerical symptom	Invariant result	Verdict
Reference	4×2	all four rows equal Z_{ref}	both invariants hold	Accept
Dropped route	4×2	$z_2 = (0.5, 0.5)$ instead of its reference row	route-count conservation broken	Reject

Shape accepts all three outcomes

Case	Shape	Numerical symptom	Invariant result	Verdict
Reference	4×2	all four rows equal Z_{ref}	both invariants hold	Accept
Dropped route	4×2	$z_2 = (0.5, 0.5)$ instead of its reference row	route-count conservation broken	Reject
Swapped origins	4×2	correct row values, rows 2 and 3 exchanged	origin-order preservation broken	Reject

Shape accepts all three outcomes

Case	Shape	Numerical symptom	Invariant result	Verdict
Reference	4×2	all four rows equal Z_{ref}	both invariants hold	Accept
Dropped route	4×2	$z_2 = (0.5, 0.5)$ instead of its reference row	route-count conservation broken	Reject
Swapped origins	4×2	correct row values, rows 2 and 3 exchanged	origin-order preservation broken	Reject

This is the chapter in one table: a shape check accepts all three outcomes, and the identity-aware checks reject two of them.

Oracles and independent references

Execution gives an observation; an oracle gives the verdict

Definition (Test oracle)
A test oracle is a predicate or metric rule $O(x, y, m) \in \{\text{accept}, \text{reject}\}$ for input x, observed output y, and interpretation metadata m. A metric oracle accepts only when a named reference, distance, and threshold satisfy $d(y, y_{\text{ref}}) \leq \tau$.

Oracle	Appropriate use
Exact predicate	token, route, index, version, membership, schema, state identity

Execution gives an observation; an oracle gives the verdict

Definition (Test oracle)
A test oracle is a predicate or metric rule $O(x, y, m) \in \{\text{accept}, \text{reject}\}$ for input x, observed output y, and interpretation metadata m. A metric oracle accepts only when a named reference, distance, and threshold satisfy $d(y, y_{\text{ref}}) \leq \tau$.

Oracle	Appropriate use
Exact predicate	token, route, index, version, membership, schema, state identity
Metric tolerance	tensor comparison under one reference, metric, dtype/shape domain, threshold

Execution gives an observation; an oracle gives the verdict

Definition (Test oracle)
A test oracle is a predicate or metric rule $O(x, y, m) \in \{\mathrm{accept}, \mathrm{reject}\}$ for input x, observed output y, and interpretation metadata m. A metric oracle accepts only when a named reference, distance, and threshold satisfy $d(y, y_{\mathrm{ref}}) \leq \tau$.

Oracle	Appropriate use
Exact predicate	token, route, index, version, membership, schema, state identity
Metric tolerance	tensor comparison under one reference, metric, dtype/shape domain, threshold
Invariant	route-count conservation, origin order, cache-position consistency

Execution gives an observation; an oracle gives the verdict

Definition (Test oracle)
A test oracle is a predicate or metric rule $O(x, y, m) \in \{\text{accept}, \text{reject}\}$ for input x, observed output y, and interpretation metadata m. A metric oracle accepts only when a named reference, distance, and threshold satisfy $d(y, y_{\text{ref}}) \leq \tau$.

Oracle	Appropriate use
Exact predicate	token, route, index, version, membership, schema, state identity
Metric tolerance	tensor comparison under one reference, metric, dtype/shape domain, threshold
Invariant	route-count conservation, origin order, cache-position consistency
Replay / conformance	a distributed trace or released interface whose verdict needs retained state

Execution gives an observation; an oracle gives the verdict

Definition (Test oracle)
A test oracle is a predicate or metric rule $O(x, y, m) \in \{\mathrm{accept}, \mathrm{reject}\}$ for input x, observed output y, and interpretation metadata m. A metric oracle accepts only when a named reference, distance, and threshold satisfy $d(y, y_{\mathrm{ref}}) \leq \tau$.

Oracle	Appropriate use
Exact predicate	token, route, index, version, membership, schema, state identity
Metric tolerance	tensor comparison under one reference, metric, dtype/shape domain, threshold
Invariant	route-count conservation, origin order, cache-position consistency
Replay / conformance	a distributed trace or released interface whose verdict needs retained state
Expected rejection	malformed routes, stale handles, invalid cache reuse, illegal transitions

Execution gives an observation; an oracle gives the verdict

Definition (Test oracle)
A test oracle is a predicate or metric rule $O(x, y, m) \in \{\text{accept}, \text{reject}\}$ for input x, observed output y, and interpretation metadata m. A metric oracle accepts only when a named reference, distance, and threshold satisfy $d(y, y_{\text{ref}}) \leq \tau$.

Oracle	Appropriate use
Exact predicate	token, route, index, version, membership, schema, state identity
Metric tolerance	tensor comparison under one reference, metric, dtype/shape domain, threshold
Invariant	route-count conservation, origin order, cache-position consistency
Replay / conformance	a distributed trace or released interface whose verdict needs retained state
Expected rejection	malformed routes, stale handles, invalid cache reuse, illegal transitions

Choose the simplest oracle that can falsify the stated behavior.

Independence is the point, not the cost

Definition (Independent reference procedure)
An independent reference procedure is a deliberately simple implementation $R : X \to Y_{\mathrm{ref}}$ whose semantics are stated independently of the production path. It should be slower, clearer, and easier to inspect.

- The MoE reference loops over origin tokens and route slots, applies F_e, and accumulates by origin id — no expert grouping, no communication.

Independence is the point, not the cost

Definition (Independent reference procedure)
An independent reference procedure is a deliberately simple implementation $R : X \to Y_{\text{ref}}$ whose semantics are stated independently of the production path. It should be slower, clearer, and easier to inspect.

- The MoE reference loops over origin tokens and route slots, applies F_e, and accumulates by origin id — no expert grouping, no communication.
- FlashMLA ships explicit PyTorch attention references; DeepGEMM exposes numerical comparison helpers; DeepEP includes reference dispatch/combine procedures. [INSPECTED: FlashMLA, DeepGEMM, DeepEP at pinned revisions]

Independence is the point, not the cost

Definition (Independent reference procedure)
An independent reference procedure is a deliberately simple implementation $R : X \to Y_{\mathrm{ref}}$ whose semantics are stated independently of the production path. It should be slower, clearer, and easier to inspect.

- The MoE reference loops over origin tokens and route slots, applies F_e, and accumulates by origin id — no expert grouping, no communication.
- FlashMLA ships explicit PyTorch attention references; DeepGEMM exposes numerical comparison helpers; DeepEP includes reference dispatch/combine procedures. [INSPECTED: FlashMLA, DeepGEMM, DeepEP at pinned revisions]
- Negative fixtures matter equally: a duplicate route slot, non-finite weight, mismatched handle, or invalid sparse index should be rejected or diagnosed, not passed through a friendly path.

Independence is the point, not the cost

Definition (Independent reference procedure)
An independent reference procedure is a deliberately simple implementation $R : X \to Y_{\text{ref}}$ whose semantics are stated independently of the production path. It should be slower, clearer, and easier to inspect.

- The MoE reference loops over origin tokens and route slots, applies F_e, and accumulates by origin id — no expert grouping, no communication.
- FlashMLA ships explicit PyTorch attention references; DeepGEMM exposes numerical comparison helpers; DeepEP includes reference dispatch/combine procedures. [INSPECTED: FlashMLA, DeepGEMM, DeepEP at pinned revisions]
- Negative fixtures matter equally: a duplicate route slot, non-finite weight, mismatched handle, or invalid sparse index should be rejected or diagnosed, not passed through a friendly path.

If both paths share an implementation, they can share a defect.

Distributed replay

Replay keeps what made the value meaningful

Replay field	Question it must answer
Topology and representation	which rank owns each object; do counts describe route slots or compact rows?

Replay keeps what made the value meaningful

Replay field	Question it must answer
Topology and representation	which rank owns each object; do counts describe route slots or compact rows?
Route and origin identity	can every received row be traced to token, slot, expert, weight, expected output?

Replay keeps what made the value meaningful

Replay field	Question it must answer
Topology and representation	which rank owns each object; do counts describe route slots or compact rows?
Route and origin identity	can every received row be traced to token, slot, expert, weight, expected output?
Handle lineage	which dispatch or layout generation produced the handle the consumer used?

Replay keeps what made the value meaningful

Replay field	Question it must answer
Topology and representation	which rank owns each object; do counts describe route slots or compact rows?
Route and origin identity	can every received row be traced to token, slot, expert, weight, expected output?
Handle lineage	which dispatch or layout generation produced the handle the consumer used?
Event / dependency trace	which sends, receives, streams, stages, and microbatches ordered the result?

Replay keeps what made the value meaningful

Replay field	Question it must answer
Topology and representation	which rank owns each object; do counts describe route slots or compact rows?
Route and origin identity	can every received row be traced to token, slot, expert, weight, expected output?
Handle lineage	which dispatch or layout generation produced the handle the consumer used?
Event / dependency trace	which sends, receives, streams, stages, and microbatches ordered the result?
Versioned content state	which layout epoch, object version, target state, manifest, checksum?

Replay keeps what made the value meaningful

Replay field	Question it must answer
Topology and representation	which rank owns each object; do counts describe route slots or compact rows?
Route and origin identity	can every received row be traced to token, slot, expert, weight, expected output?
Handle lineage	which dispatch or layout generation produced the handle the consumer used?
Event / dependency trace	which sends, receives, streams, stages, and microbatches ordered the result?
Versioned content state	which layout epoch, object version, target state, manifest, checksum?

A replay record extends ordinary run metadata with exactly these fields — the smallest artifact set sufficient to rerun or inspect a distributed failure.

A negative trace where every field is plausible

For request row ρ_7, route slot 0, logical expert 6, weight 0.65, one placement generation G permits ranks 2 and 6 and selects rank 2:

$$\mathsf{dispatch}_G(\rho_7, 0, 6, 2) \prec \mathsf{complete}_G(2) \prec \mathsf{combine}_G(\rho_7, 0).$$

A negative trace where every field is plausible

For request row ρ_7, route slot 0, logical expert 6, weight 0.65, one placement generation G permits ranks 2 and 6 and selects rank 2:

$$\mathsf{dispatch}_G(\rho_7, 0, 6, 2) \prec \mathsf{complete}_G(2) \prec \mathsf{combine}_G(\rho_7, 0).$$

The negative trace selects rank 6 under a newer generation while reusing rank-2 handle and completion state. Every field is plausible; the combination is an illegal handoff.

What the reviewed replays established — and did not

Established

- A two-rank application guard rejected aliased route mutation before native DeepEP dispatch; copied cached handles stayed isolated.
- A four-H200 DeepEP/EPLB closure passed the recorded ordinary, cached, expanded, FP8, combine, and reduced-combine assertions, plus eight EPLB replays.
- A four-H200 DualPipe callback replay passed its sequential reference in serialized and concurrent modes.

Still open

- Native DeepEP stale-handle detection.
- Universal forward/input-gradient overlap: the profiler showed it in *four of twelve* callback windows.
- Speedup and production engine behavior.

[OBSERVED: reviewed July 21 H200 closure and callback replays] [OPEN: native stale-handle detection, universal overlap]

What the reviewed replays established — and did not

Established

- A two-rank application guard rejected aliased route mutation before native DeepEP dispatch; copied cached handles stayed isolated.
- A four-H200 DeepEP/EPLB closure passed the recorded ordinary, cached, expanded, FP8, combine, and reduced-combine assertions, plus eight EPLB replays.
- A four-H200 DualPipe callback replay passed its sequential reference in serialized and concurrent modes.

Still open

- Native DeepEP stale-handle detection.
- Universal forward/input-gradient overlap: the profiler showed it in *four of twelve* callback windows.
- Speedup and production engine behavior.

[OBSERVED: reviewed July 21 H200 closure and callback replays] [OPEN: native stale-handle detection, universal overlap]

Storage reuses the same rule with different identities: table version decodes layout, chain version validates routing, chunk version identifies content.

Cross-surface handoffs

A threshold is not portable without its record

Definition (Tolerance record)
A tolerance record is $T = (R, d, \tau, D, E)$: reference procedure R, metric d, threshold τ, dtype/shape domain D, and pinned revision and runtime environment E.

Suppose a low-precision expert path returns $\widehat{z}_3 = (3.69, -2.72)$ instead of $z_3 = (3.7, -2.7)$. Then

$$\|\widehat{z}_3 - z_3\|_\infty = 0.02.$$

- Under a recorded threshold 0.025: the row passes.

A threshold is not portable without its record

Definition (Tolerance record)
A tolerance record is $T = (R, d, \tau, D, E)$: reference procedure R, metric d, threshold τ, dtype/shape domain D, and pinned revision and runtime environment E.

Suppose a low-precision expert path returns $\widehat{z}_3 = (3.69, -2.72)$ instead of $z_3 = (3.7, -2.7)$. Then

$$\|\widehat{z}_3 - z_3\|_\infty = 0.02.$$

- Under a recorded threshold 0.025: the row passes.
- Under 0.01: the same fixed observation fails.

A threshold is not portable without its record

Definition (Tolerance record)
A tolerance record is $T = (R, d, \tau, D, E)$: reference procedure R, metric d, threshold τ, dtype/shape domain D, and pinned revision and runtime environment E.

Suppose a low-precision expert path returns $\widehat{z}_3 = (3.69, -2.72)$ instead of $z_3 = (3.7, -2.7)$. Then

$$\|\widehat{z}_3 - z_3\|_\infty = 0.02.$$

- Under a recorded threshold 0.025: the row passes.
- Under 0.01: the same fixed observation fails.
- Exact route, scale-block, sparse-index, and inverse-map checks remain mandatory either way.

A threshold is not portable without its record

Definition (Tolerance record)
A tolerance record is $T = (R, d, \tau, D, E)$: reference procedure R, metric d, threshold τ, dtype/shape domain D, and pinned revision and runtime environment E.

Suppose a low-precision expert path returns $\widehat{z}_3 = (3.69, -2.72)$ instead of $z_3 = (3.7, -2.7)$. Then

$$\|\widehat{z}_3 - z_3\|_\infty = 0.02.$$

- Under a recorded threshold 0.025: the row passes.
- Under 0.01: the same fixed observation fails.
- Exact route, scale-block, sparse-index, and inverse-map checks remain mandatory either way.

One observation, two verdicts. The number means nothing until R, d, τ, D, and E are all fixed.

From concept to code: the metric is a named function

A tolerance record's d is not a vague notion of "close." In DeepGEMM it is a specific similarity-based scalar, computed in double precision.

from **DeepGEMM/deep_gemm/testing/numeric.py**

```
def calc_diff(x: torch.Tensor, y: torch.Tensor):
    x, y = x.double(), y.double()
    denominator = (x * x + y * y).sum()
    ...
    sim = 2 * (x * y).sum() / denominator
    return 1 - sim
```

From concept to code: the metric is a named function

A tolerance record's d is not a vague notion of "close." In DeepGEMM it is a specific similarity-based scalar, computed in double precision.

from **DeepGEMM/deep_gemm/testing/numeric.py**

```
def calc_diff(x: torch.Tensor, y: torch.Tensor):
    x, y = x.double(), y.double()
    denominator = (x * x + y * y).sum()
    ...
    sim = 2 * (x * y).sum() / denominator
    return 1 - sim
```

The same module also ships `assert_bitwise_equal`, which reports the first mismatching byte, element, and coordinate. Those are the two ends of the taxonomy — a metric oracle and an exact predicate — in one file. [INSPECTED: DeepGEMM `deep_gemm/testing/numeric.py`]

Repository interfaces and protocol scope

Three test modes, three bounded conclusions

Test mode	Execution and bounded conclusion
Reference test	Concrete repository code runs selected fixtures. The implementation trace passed the named exact, invariant, or numerical oracle.

Three test modes, three bounded conclusions

Test mode	Execution and bounded conclusion
Reference test	Concrete repository code runs selected fixtures. The implementation trace passed the named exact, invariant, or numerical oracle.
Protocol model checking	An abstract model runs generated events and failures within configured bounds. Enabled monitors held for explored schedules; an abstraction map connects them to production code.

Three test modes, three bounded conclusions

Test mode	Execution and bounded conclusion
Reference test	Concrete repository code runs selected fixtures. The implementation trace passed the named exact, invariant, or numerical oracle.
Protocol model checking	An abstract model runs generated events and failures within configured bounds. Enabled monitors held for explored schedules; an abstraction map connects them to production code.
Open integration test	A runtime invokes a released interface through a pinned adapter. The exercised path preserved the named interface; underlying operators require their own oracles.

Three test modes, three bounded conclusions

Test mode	Execution and bounded conclusion
Reference test	Concrete repository code runs selected fixtures. The implementation trace passed the named exact, invariant, or numerical oracle.
Protocol model checking	An abstract model runs generated events and failures within configured bounds. Enabled monitors held for explored schedules; an abstraction map connects them to production code.
Open integration test	A runtime invokes a released interface through a pinned adapter. The exercised path preserved the named interface; underlying operators require their own oracles.

Definition (Abstraction and refinement boundary)

With concrete C++ traces C, abstract P traces A, and abstraction map $\alpha : C \to A$, model checking establishes $P(a)$ only for explored $a \in A$. A production conclusion additionally needs $c \in C_{\text{scope}} \Rightarrow \alpha(c) \in A$ and $P(\alpha(c))$.

Three test modes, three bounded conclusions

Test mode	Execution and bounded conclusion
Reference test	Concrete repository code runs selected fixtures. The implementation trace passed the named exact, invariant, or numerical oracle.
Protocol model checking	An abstract model runs generated events and failures within configured bounds. Enabled monitors held for explored schedules; an abstraction map connects them to production code.
Open integration test	A runtime invokes a released interface through a pinned adapter. The exercised path preserved the named interface; underlying operators require their own oracles.

Definition (Abstraction and refinement boundary)
With concrete C++ traces C, abstract P traces A, and abstraction map $\alpha : C \to A$, model checking establishes $P(a)$ only for explored $a \in A$. A production conclusion additionally needs $c \in C_{\text{scope}} \Rightarrow \alpha(c) \in A$ and $P(\alpha(c))$.

The pinned repository supplies no general C++ refinement proof. [INSPECTED: 3FS P specifications at the pinned revision]

Coverage rows use different denominators

Subsystem	Coverage actually available
Kernels	sampled standalone H200 rows passed, including an 18-row July 19 replication; whole-runtime behavior open

Coverage rows use different denominators

Subsystem	Coverage actually available
Kernels	sampled standalone H200 rows passed, including an 18-row July 19 replication; whole-runtime behavior open
DeepEP / EPLB / DualPipe	reviewed four-H200 assertions, a two-rank caller guard, eight EPLB replays, sampled-reference callback checks

Coverage rows use different denominators

Subsystem	Coverage actually available
Kernels	sampled standalone H200 rows passed, including an 18-row July 19 replication; whole-runtime behavior open
DeepEP / EPLB / DualPipe	reviewed four-H200 assertions, a two-rank caller guard, eight EPLB replays, sampled-reference callback checks
3FS concrete path	source-history fixes define candidate regressions; deployment-backed replay and verified restore open

Coverage rows use different denominators

Subsystem	Coverage actually available
Kernels	sampled standalone H200 rows passed, including an 18-row July 19 replication; whole-runtime behavior open
DeepEP / EPLB / DualPipe	reviewed four-H200 assertions, a two-rank caller guard, eight EPLB replays, sampled-reference callback checks
3FS concrete path	source-history fixes define candidate regressions; deployment-backed replay and verified restore open
3FS P specifications	enabled monitors within bounded schedules; inactive replica-content assertions and general refinement unproved

Coverage rows use different denominators

Subsystem	Coverage actually available
Kernels	sampled standalone H200 rows passed, including an 18-row July 19 replication; whole-runtime behavior open
DeepEP / EPLB / DualPipe	reviewed four-H200 assertions, a two-rank caller guard, eight EPLB replays, sampled-reference callback checks
3FS concrete path	source-history fixes define candidate regressions; deployment-backed replay and verified restore open
3FS P specifications	enabled monitors within bounded schedules; inactive replica-content assertions and general refinement unproved
Long-context cache	one reviewed four-H200 2K-prefix probe passed six prefix-cache oracles; indexer and quality cases open

Coverage rows use different denominators

Subsystem	Coverage actually available
Kernels	sampled standalone H200 rows passed, including an 18-row July 19 replication; whole-runtime behavior open
DeepEP / EPLB / DualPipe	reviewed four-H200 assertions, a two-rank caller guard, eight EPLB replays, sampled-reference callback checks
3FS concrete path	source-history fixes define candidate regressions; deployment-backed replay and verified restore open
3FS P specifications	enabled monitors within bounded schedules; inactive replica-content assertions and general refinement unproved
Long-context cache	one reviewed four-H200 2K-prefix probe passed six prefix-cache oracles; indexer and quality cases open
Open-runtime adapter	the RTX 5090 differential changed attention *and* sampling, so it is sensitivity, not conformance

Coverage rows use different denominators

Subsystem	Coverage actually available
Kernels	sampled standalone H200 rows passed, including an 18-row July 19 replication; whole-runtime behavior open
DeepEP / EPLB / DualPipe	reviewed four-H200 assertions, a two-rank caller guard, eight EPLB replays, sampled-reference callback checks
3FS concrete path	source-history fixes define candidate regressions; deployment-backed replay and verified restore open
3FS P specifications	enabled monitors within bounded schedules; inactive replica-content assertions and general refinement unproved
Long-context cache	one reviewed four-H200 2K-prefix probe passed six prefix-cache oracles; indexer and quality cases open
Open-runtime adapter	the RTX 5090 differential changed attention *and* sampling, so it is sensitivity, not conformance

A cache-layout change forces new index, backend, memory, and long-context checks. A placement change forces new dispatch, combine, and replay checks. A storage-version change forces new publication and restore checks.

[OBSERVED: reviewed H200 packets] [OPEN: deployment correctness]

Concept-to-code map for the chapter

- **attention reference with explicit mask and log-sum-exp** → `FlashMLA/tests/ref.py`

Concept-to-code map for the chapter

- **attention reference with explicit mask and log-sum-exp** → `FlashMLA/tests/ref.py`
- **similarity metric and bitwise predicate** → `DeepGEMM/deep_gemm/testing/numeric.py`

Concept-to-code map for the chapter

- **attention reference with explicit mask and log-sum-exp** → `FlashMLA/tests/ref.py`
- **similarity metric and bitwise predicate** → `DeepGEMM/deep_gemm/testing/numeric.py`
- **reference dispatch/combine with origin identity** → `DeepEP/deep_ep/utils/refs.py`

Concept-to-code map for the chapter

- **attention reference with explicit mask and log-sum-exp** → `FlashMLA/tests/ref.py`
- **similarity metric and bitwise predicate** → `DeepGEMM/deep_gemm/testing/numeric.py`
- **reference dispatch/combine with origin identity** → `DeepEP/deep_ep/utils/refs.py`
- **inverse-map and invalid-slot assertions** → `TileKernels/tests/moe/test_get_fused_mapping.py`

Concept-to-code map for the chapter

- **attention reference with explicit mask and log-sum-exp** → `FlashMLA/tests/ref.py`
- **similarity metric and bitwise predicate** → `DeepGEMM/deep_gemm/testing/numeric.py`
- **reference dispatch/combine with origin identity** → `DeepEP/deep_ep/utils/refs.py`
- **inverse-map and invalid-slot assertions** → `TileKernels/tests/moe/test_get_fused_mapping.py`
- **abstract storage protocol and monitors** → `3FS/specs/DataStorage/PSpec/SystemSpec.p`

Concept-to-code map for the chapter

- **attention reference with explicit mask and log-sum-exp** → `FlashMLA/tests/ref.py`
- **similarity metric and bitwise predicate** → `DeepGEMM/deep_gemm/testing/numeric.py`
- **reference dispatch/combine with origin identity** → `DeepEP/deep_ep/utils/refs.py`
- **inverse-map and invalid-slot assertions** → `TileKernels/tests/moe/test_get_fused_mapping.py`
- **abstract storage protocol and monitors** → `3FS/specs/DataStorage/PSpec/SystemSpec.p`
- **versioned content state** → `3FS/src/fbs/storage/Common.h`

Concept-to-code map for the chapter

- **attention reference with explicit mask and log-sum-exp** → `FlashMLA/tests/ref.py`
- **similarity metric and bitwise predicate** → `DeepGEMM/deep_gemm/testing/numeric.py`
- **reference dispatch/combine with origin identity** → `DeepEP/deep_ep/utils/refs.py`
- **inverse-map and invalid-slot assertions** → `TileKernels/tests/moe/test_get_fused_mapping.py`
- **abstract storage protocol and monitors** → `3FS/specs/DataStorage/PSpec/SystemSpec.p`
- **versioned content state** → `3FS/src/fbs/storage/Common.h`
- **chain-table version advance** → `3FS/src/mgmtd/ops/SetChainTableOperation.cc`

Concept-to-code map for the chapter

- **attention reference with explicit mask and log-sum-exp** → `FlashMLA/tests/ref.py`
- **similarity metric and bitwise predicate** → `DeepGEMM/deep_gemm/testing/numeric.py`
- **reference dispatch/combine with origin identity** → `DeepEP/deep_ep/utils/refs.py`
- **inverse-map and invalid-slot assertions** → `TileKernels/tests/moe/test_get_fused_mapping.py`
- **abstract storage protocol and monitors** → `3FS/specs/DataStorage/PSpec/SystemSpec.p`
- **versioned content state** → `3FS/src/fbs/storage/Common.h`
- **chain-table version advance** → `3FS/src/mgmtd/ops/SetChainTableOperation.cc`

Name the object, state what must be preserved, choose the oracle, retain the smallest trace that can explain a failure.

Next: report this verdict alongside task quality, performance, cost and capacity, safety, and operability — the volume conclusion combines them into a release-fitness view.