DeepSeekMoE and Sparse Computation

Deep Dive into DeepSeek — Chapter 3

Yin Yang

Foundation Books Project

The one rule of this chapter

A sparse layer may change *which work happens*.
It may not change the output type or the token's identity.

- The surrounding transformer block receives a hidden state of the original width — not an expert id, a route table, or a packed buffer.
- So sparsity buys active-compute savings and pays for them in *metadata*: expert ids, route weights, dispatch positions, combine destinations.
- That metadata is the route record. It is the object this chapter builds, and the object every later systems layer inherits.

The one rule of this chapter

A sparse layer may change *which work happens*.
It may not change the output type or the token's identity.

- The surrounding transformer block receives a hidden state of the original width — not an expert id, a route table, or a packed buffer.
- So sparsity buys active-compute savings and pays for them in *metadata*: expert ids, route weights, dispatch positions, combine destinations.
- That metadata is the route record. It is the object this chapter builds, and the object every later systems layer inherits.

Goal: follow four tokens through one MoE layer — scores, top-k, weights, capacity, group limits, balance, dispatch, combine — and leave with a route specification that a distributed runtime can be tested against.

Roadmap

1. Sparse computation as a cost decision

Roadmap

1. Sparse computation as a cost decision
2. The route record

Roadmap

1. Sparse computation as a cost decision
2. The route record
3. Selection, weights, and capacity

Roadmap

1. Sparse computation as a cost decision
2. The route record
3. Selection, weights, and capacity
4. Group limits and load balance

Roadmap

1. Sparse computation as a cost decision
2. The route record
3. Selection, weights, and capacity
4. Group limits and load balance
5. Dispatch and combine

Roadmap

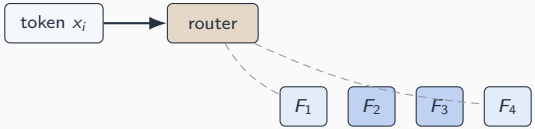
1. Sparse computation as a cost decision
2. The route record
3. Selection, weights, and capacity
4. Group limits and load balance
5. Dispatch and combine
6. Failure modes and what comes next

Sparse computation as a cost decision

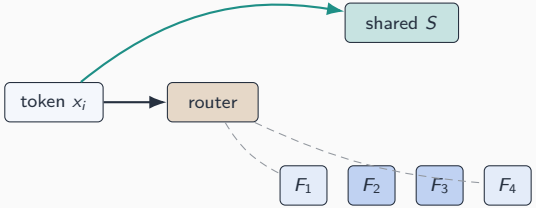
Stored capacity is not per-token work



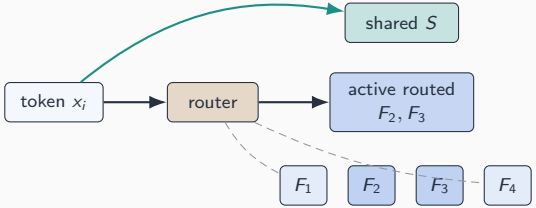
Stored capacity is not per-token work



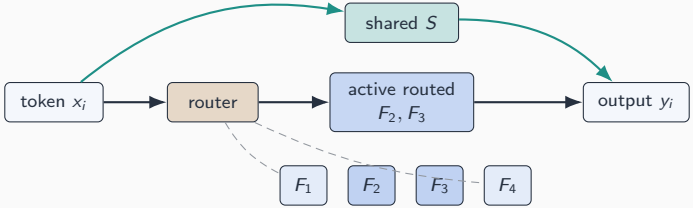
Stored capacity is not per-token work



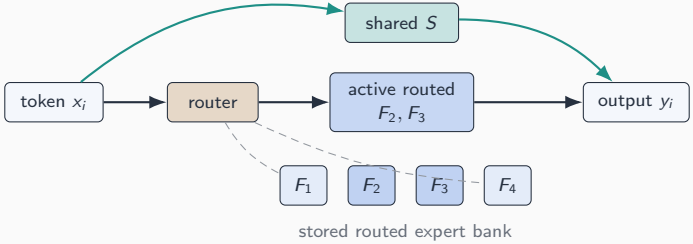
Stored capacity is not per-token work



Stored capacity is not per-token work

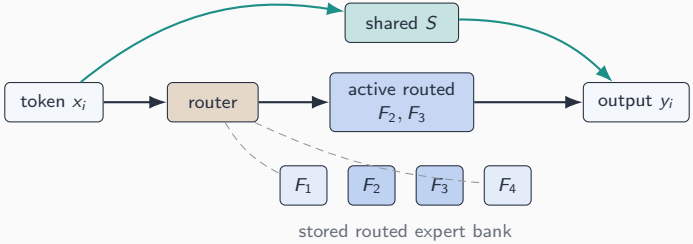


Stored capacity is not per-token work



- A dense feed-forward network applies the *same* weights to every token. A sparse expert layer stores many parameter sets and routes each token through a few.
- That separates total model capacity from active per-token work.

Stored capacity is not per-token work



- A dense feed-forward network applies the *same* weights to every token. A sparse expert layer stores many parameter sets and routes each token through a few.
- That separates total model capacity from active per-token work.

And it creates state a dense block never needs: expert ids, route weights, dispatch positions, combine destinations.

Two rows: the real one and the checkable one

Quantity	DeepSeek-V3 (671B config)	Chapter's toy layer
routed experts	256	4

[INSPECTED: DeepSeek-V3 `inference/configs/config_671B.json`]

Two rows: the real one and the checkable one

Quantity	DeepSeek-V3 (671B config)	Chapter's toy layer
routed experts	256	4
shared experts	1	1

[INSPECTED: DeepSeek-V3 `inference/configs/config_671B.json`]

Two rows: the real one and the checkable one

Quantity	DeepSeek-V3 (671B config)	Chapter's toy layer
routed experts	256	4
shared experts	1	1
activated routed experts per token	8	2

[INSPECTED: DeepSeek-V3 `inference/configs/config_671B.json`]

Two rows: the real one and the checkable one

Quantity	DeepSeek-V3 (671B config)	Chapter's toy layer
routed experts	256	4
shared experts	1	1
activated routed experts per token	8	2
expert groups / limited groups	8 / 4	2 / 1

[INSPECTED: DeepSeek-V3 `inference/configs/config_671B.json`]

Two rows: the real one and the checkable one

Quantity	DeepSeek-V3 (671B config)	Chapter's toy layer
routed experts	256	4
shared experts	1	1
activated routed experts per token	8	2
expert groups / limited groups	8 / 4	2 / 1
score transform, route scale	sigmoid, 2.5	normalize to 1

[INSPECTED: DeepSeek-V3 `inference/configs/config_671B.json`]

Two rows: the real one and the checkable one

Quantity	DeepSeek-V3 (671B config)	Chapter's toy layer
routed experts	256	4
shared experts	1	1
activated routed experts per token	8	2
expert groups / limited groups	8 / 4	2 / 1
score transform, route scale	sigmoid, 2.5	normalize to 1

[INSPECTED: DeepSeek-V3 `inference/configs/config_671B.json`]

- The left column is the source row the chapter is really about; the right column is the row you can audit by hand.
- For V3, 256 routed experts are stored capacity and 8 activated routed experts are per-token work — neither is the block's output width.

Two rows: the real one and the checkable one

Quantity	DeepSeek-V3 (671B config)	Chapter's toy layer
routed experts	256	4
shared experts	1	1
activated routed experts per token	8	2
expert groups / limited groups	8 / 4	2 / 1
score transform, route scale	sigmoid, 2.5	normalize to 1

[INSPECTED: DeepSeek-V3 `inference/configs/config_671B.json`]

- The left column is the source row the chapter is really about; the right column is the row you can audit by hand.
- For V3, 256 routed experts are stored capacity and 8 activated routed experts are per-token work — neither is the block's output width.

Four tokens, four routed experts, one shared expert, top-2. Every number in this chapter can be checked with a pencil.

The route record

What the formula leaves out

$$M(x_i) \;=\; S(x_i) \;+\; \sum_{e \in T_i} w_{i,e}\, F_e(x_i), \qquad F_e, S : \mathbb{R}^d \to \mathbb{R}^d$$

The formula needs a *set* T_i and coefficients $w_{i,e}$. A program needs more:

Field	Mathematical role	Why an implementation needs it
i, p_i	token identity, original position	restores batch/sequence order

What the formula leaves out

$$M(x_i) \;=\; S(x_i) \;+\; \sum_{e \in T_i} w_{i,e}\, F_e(x_i), \qquad F_e, S : \mathbb{R}^d \to \mathbb{R}^d$$

The formula needs a *set* T_i and coefficients $w_{i,e}$. A program needs more:

Field	Mathematical role	Why an implementation needs it
i, p_i	token identity, original position	restores batch/sequence order
r_i	score vector	makes selection and tie-break evidence explicit

What the formula leaves out

$$M(x_i) \;=\; S(x_i) \;+\; \sum_{e \in T_i} w_{i,e}\, F_e(x_i), \qquad F_e, S : \mathbb{R}^d \to \mathbb{R}^d$$

The formula needs a *set* T_i and coefficients $w_{i,e}$. A program needs more:

Field	Mathematical role	Why an implementation needs it
i, p_i	token identity, original position	restores batch/sequence order
r_i	score vector	makes selection and tie-break evidence explicit
e, q	expert id, selection rank	picks F_e; keeps top-k columns aligned

What the formula leaves out

$$M(x_i) \;=\; S(x_i) \;+\; \sum_{e \in T_i} w_{i,e}\, F_e(x_i), \qquad F_e, S : \mathbb{R}^d \to \mathbb{R}^d$$

The formula needs a *set* T_i and coefficients $w_{i,e}$. A program needs more:

Field	Mathematical role	Why an implementation needs it
i, p_i	token identity, original position	restores batch/sequence order
r_i	score vector	makes selection and tie-break evidence explicit
e, q	expert id, selection rank	picks F_e; keeps top-k columns aligned
$w_{i,e}$	route weight	scales that expert's output at combine

What the formula leaves out

$$M(x_i) \;=\; S(x_i) \;+\; \sum_{e \in T_i} w_{i,e}\, F_e(x_i), \qquad F_e, S : \mathbb{R}^d \to \mathbb{R}^d$$

The formula needs a *set* T_i and coefficients $w_{i,e}$. A program needs more:

Field	Mathematical role	Why an implementation needs it
i, p_i	token identity, original position	restores batch/sequence order
r_i	score vector	makes selection and tie-break evidence explicit
e, q	expert id, selection rank	picks F_e; keeps top-k columns aligned
$w_{i,e}$	route weight	scales that expert's output at combine
s_{dispatch}, s_{combine}	packed row, return row	finds and returns the token

What the formula leaves out

$$M(x_i) \;=\; S(x_i) \;+\; \sum_{e \in T_i} w_{i,e}\, F_e(x_i), \qquad F_e, S : \mathbb{R}^d \to \mathbb{R}^d$$

The formula needs a *set* T_i and coefficients $w_{i,e}$. A program needs more:

Field	Mathematical role	Why an implementation needs it
i, p_i	token identity, original position	restores batch/sequence order
r_i	score vector	makes selection and tie-break evidence explicit
e, q	expert id, selection rank	picks F_e; keeps top-k columns aligned
$w_{i,e}$	route weight	scales that expert's output at combine
s_{dispatch}, s_{combine}	packed row, return row	finds and returns the token

That tuple is the router record. Dropping any field can still produce shape-correct buffers whose rows are attributed to the wrong token or expert.

Fine-grained segmentation and shared expert isolation

DeepSeekMoE move	What changes in the layer	Technical consequence
Fine-grained segmentation	routed capacity is split into more, smaller expert units	route records name a precise subset of specialists, not one coarse expert

[REPORTED: DeepSeekMoE paper]

Fine-grained segmentation and shared expert isolation

DeepSeekMoE move	What changes in the layer	Technical consequence
Fine-grained segmentation	routed capacity is split into more, smaller expert units	route records name a precise subset of specialists, not one coarse expert
Shared expert isolation	common computation is placed on an always-active shared path	active-parameter accounting includes shared work *plus* routed work

[REPORTED: DeepSeekMoE paper]

Fine-grained segmentation and shared expert isolation

DeepSeekMoE move	What changes in the layer	Technical consequence
Fine-grained segmentation	routed capacity is split into more, smaller expert units	route records name a precise subset of specialists, not one coarse expert
Shared expert isolation	common computation is placed on an always-active shared path	active-parameter accounting includes shared work *plus* routed work

[REPORTED: DeepSeekMoE paper]

- "Common knowledge" here names generally useful learned computation. A shared expert is not an Engram-style lookup table.

Fine-grained segmentation and shared expert isolation

DeepSeekMoE move	What changes in the layer	Technical consequence
Fine-grained segmentation	routed capacity is split into more, smaller expert units	route records name a precise subset of specialists, not one coarse expert
Shared expert isolation	common computation is placed on an always-active shared path	active-parameter accounting includes shared work *plus* routed work

[REPORTED: DeepSeekMoE paper]

- "Common knowledge" here names generally useful learned computation. A shared expert is not an Engram-style lookup table.
- One shared expert plus top-2 routed selection means each token evaluates *three* expert functions, not two.

Selection, weights, and capacity

Four tokens, top-2, one deterministic tie-break

Token	$r_{i,1}$	$r_{i,2}$	$r_{i,3}$	$r_{i,4}$	Selected	Weights
x_1	0.10	0.70	0.15	0.05	$2, 3$	$14/17,\ 3/17$

[DERIVED: chapter's normalization rule on the toy score matrix]

Four tokens, top-2, one deterministic tie-break

Token	$r_{i,1}$	$r_{i,2}$	$r_{i,3}$	$r_{i,4}$	Selected	Weights
x_1	0.10	0.70	0.15	0.05	$2, 3$	$14/17,\ 3/17$
x_2	0.50	0.20	0.10	0.20	$1, 2$	$5/7,\ 2/7$

[DERIVED: chapter's normalization rule on the toy score matrix]

Four tokens, top-2, one deterministic tie-break

Token	$r_{i,1}$	$r_{i,2}$	$r_{i,3}$	$r_{i,4}$	Selected	Weights
x_1	0.10	0.70	0.15	0.05	$2, 3$	$14/17,\ 3/17$
x_2	0.50	0.20	0.10	0.20	$1, 2$	$5/7,\ 2/7$
x_3	0.05	0.15	0.35	0.45	$4, 3$	$9/16,\ 7/16$

[DERIVED: chapter's normalization rule on the toy score matrix]

Four tokens, top-2, one deterministic tie-break

Token	$r_{i,1}$	$r_{i,2}$	$r_{i,3}$	$r_{i,4}$	Selected	Weights
x_1	0.10	0.70	0.15	0.05	$2, 3$	$14/17,\ 3/17$
x_2	0.50	0.20	0.10	0.20	$1, 2$	$5/7,\ 2/7$
x_3	0.05	0.15	0.35	0.45	$4, 3$	$9/16,\ 7/16$
x_4	0.40	0.30	0.20	0.10	$1, 2$	$4/7,\ 3/7$

[DERIVED: chapter's normalization rule on the toy score matrix]

Four tokens, top-2, one deterministic tie-break

Token	$r_{i,1}$	$r_{i,2}$	$r_{i,3}$	$r_{i,4}$	Selected	Weights
x_1	0.10	0.70	0.15	0.05	$2, 3$	$14/17,\ 3/17$
x_2	0.50	0.20	0.10	0.20	$1, 2$	$5/7,\ 2/7$
x_3	0.05	0.15	0.35	0.45	$4, 3$	$9/16,\ 7/16$
x_4	0.40	0.30	0.20	0.10	$1, 2$	$4/7,\ 3/7$

[DERIVED: chapter's normalization rule on the toy score matrix]

- Weights normalize the *selected* scores: $w_{i,e} = r_{i,e} / \sum_{j \in T_i} r_{i,j}$. Transparent arithmetic, not V3's rule.
- Row x_2 ties at 0.20 between experts 2 and 4; the smaller expert id wins, so the example stays deterministic.

Four tokens, top-2, one deterministic tie-break

Token	$r_{i,1}$	$r_{i,2}$	$r_{i,3}$	$r_{i,4}$	Selected	Weights
x_1	0.10	0.70	0.15	0.05	$2, 3$	$14/17,\ 3/17$
x_2	0.50	0.20	0.10	0.20	$1, 2$	$5/7,\ 2/7$
x_3	0.05	0.15	0.35	0.45	$4, 3$	$9/16,\ 7/16$
x_4	0.40	0.30	0.20	0.10	$1, 2$	$4/7,\ 3/7$

[DERIVED: chapter's normalization rule on the toy score matrix]

- Weights normalize the *selected* scores: $w_{i,e} = r_{i,e} / \sum_{j \in T_i} r_{i,j}$. Transparent arithmetic, not V3's rule.
- Row x_2 ties at 0.20 between experts 2 and 4; the smaller expert id wins, so the example stays deterministic.

Route-entry load: $c = (2, 3, 2, 1)$. Expert 2 is already hot relative to the batch mean of 2.

Capacity counts route entries, not tokens

Definition (Expert capacity)
For expert e, let $c_e = |\{(i, e) : e \in T_i\}|$ be the selected route entries sent to that expert in a batch. A capacity C_e requires $c_e \leq C_e$, unless the implementation specifies an overflow policy.

- Set $C_e = 2$: expert 2 overflows, because $c_2 = 3$. A token selected by two experts contributes *two* route entries.

Capacity counts route entries, not tokens

Definition (Expert capacity)
For expert e, let $c_e = |\{(i, e) : e \in T_i\}|$ be the selected route entries sent to that expert in a batch. A capacity C_e requires $c_e \leq C_e$, unless the implementation specifies an overflow policy.

- Set $C_e = 2$: expert 2 overflows, because $c_2 = 3$. A token selected by two experts contributes *two* route entries.
- Under a keep-by-packing-order policy, (x_4, F_2) with weight $3/7$ is the single overflow, and accepted loads become $(2, 2, 2, 1)$. [DERIVED: chapter checkpoint `moe_routing_capacity_checkpoint.py`]

Capacity counts route entries, not tokens

Definition (Expert capacity)
For expert e, let $c_e = |\{(i, e) : e \in T_i\}|$ be the selected route entries sent to that expert in a batch. A capacity C_e requires $c_e \leq C_e$, unless the implementation specifies an overflow policy.

- Set $C_e = 2$: expert 2 overflows, because $c_2 = 3$. A token selected by two experts contributes *two* route entries.
- Under a keep-by-packing-order policy, (x_4, F_2) with weight $3/7$ is the single overflow, and accepted loads become $(2, 2, 2, 1)$. [DERIVED: chapter checkpoint `moe_routing_capacity_checkpoint.py`]

Drop, reroute, split the batch, or raise capacity — but name the policy. Top-k alone does not determine the layer output under pressure.

Group limits and load balance

Eligibility comes before selection

- Partition the routed experts into groups. A group-limited router first picks eligible *groups*, then runs top-k inside them only.

Eligibility comes before selection

- Partition the routed experts into groups. A group-limited router first picks eligible *groups*, then runs top-k inside them only.
- Toy case $G_1 = \{1, 2\}$, $G_2 = \{3, 4\}$, one eligible group: token x_1's best group is G_1 (max 0.70 vs 0.15), so it selects $2, 1$ — not $2, 3$.

Eligibility comes before selection

- Partition the routed experts into groups. A group-limited router first picks eligible *groups*, then runs top-k inside them only.
- Toy case $G_1 = \{1, 2\}$, $G_2 = \{3, 4\}$, one eligible group: token x_1's best group is G_1 (max 0.70 vs 0.15), so it selects $2, 1$ — not $2, 3$.
- Expert 3 had the second-largest *global* score and was still ineligible. Group limits change the candidate set, not a post-hoc placement label.

Eligibility comes before selection

- Partition the routed experts into groups. A group-limited router first picks eligible *groups*, then runs top-k inside them only.
- Toy case $G_1 = \{1, 2\}$, $G_2 = \{3, 4\}$, one eligible group: token x_1's best group is G_1 (max 0.70 vs 0.15), so it selects $2, 1$ — not $2, 3$.
- Expert 3 had the second-largest *global* score and was still ineligible. Group limits change the candidate set, not a post-hoc placement label.

V3's public inference configuration records `n_expert_groups = 8` and `n_limited_groups = 4`, and its gate computes group scores, masks non-selected groups, then applies top-k.

[INSPECTED: DeepSeek-V3 `inference/model.py`, `config_671B.json`]

Eligibility comes before selection

- Partition the routed experts into groups. A group-limited router first picks eligible *groups*, then runs top-k inside them only.
- Toy case $G_1 = \{1, 2\}$, $G_2 = \{3, 4\}$, one eligible group: token x_1's best group is G_1 (max 0.70 vs 0.15), so it selects $2, 1$ — not $2, 3$.
- Expert 3 had the second-largest *global* score and was still ineligible. Group limits change the candidate set, not a post-hoc placement label.

V3's public inference configuration records `n_expert_groups = 8` and `n_limited_groups = 4`, and its gate computes group scores, masks non-selected groups, then applies top-k.

[INSPECTED: DeepSeek-V3 `inference/model.py`, `config_671B.json`]

Whether that logical restriction becomes a communication saving is a placement question — EPLB's, not the router's.

Two balance mechanisms, two paths

	Auxiliary load-balance loss	Correction bias (loss-free)
Where it acts	term λL_{bal} inside the training objective	selection offset b_i updated outside back-propagation

$$b_i^{(u+1)} = b_i^{(u)} - \gamma \, \mathrm{sgn}\left(c_i^{(u)} - \bar{c}^{(u)}\right)$$

Two balance mechanisms, two paths

	Auxiliary load-balance loss	Correction bias (loss-free)
Where it acts	term λL_{bal} inside the training objective	selection offset b_i updated outside back-propagation
What it changes	learned router parameters, via the optimizer	which experts enter T_t at the next step

$$b_i^{(u+1)} = b_i^{(u)} - \gamma \, \mathrm{sgn}\left(c_i^{(u)} - \bar{c}^{(u)}\right)$$

Two balance mechanisms, two paths

	Auxiliary load-balance loss	Correction bias (loss-free)
Where it acts	term λL_{bal} inside the training objective	selection offset b_i updated outside back-propagation
What it changes	learned router parameters, via the optimizer	which experts enter T_t at the next step
Combine weights	unaffected as a term	*also* unaffected — weights come from the unadjusted scores

$$b_i^{(u+1)} = b_i^{(u)} - \gamma \, \mathrm{sgn}\left(c_i^{(u)} - \bar{c}^{(u)}\right)$$

Two balance mechanisms, two paths

	Auxiliary load-balance loss	Correction bias (loss-free)
Where it acts	term λL_{bal} inside the training objective	selection offset b_i updated outside back-propagation
What it changes	learned router parameters, via the optimizer	which experts enter T_t at the next step
Combine weights	unaffected as a term	*also* unaffected — weights come from the unadjusted scores

$$b_i^{(u+1)} = b_i^{(u)} - \gamma \, \operatorname{sgn}\left(c_i^{(u)} - \bar{c}^{(u)}\right)$$

DeepSeek-V3 reports a load-balancing strategy without a batch-wise auxiliary loss, plus one very small complementary *sequence-wise* term. [REPORTED: DeepSeek-V3 technical report]

Two balance mechanisms, two paths

	Auxiliary load-balance loss	Correction bias (loss-free)
Where it acts	term λL_{bal} inside the training objective	selection offset b_i updated outside back-propagation
What it changes	learned router parameters, via the optimizer	which experts enter T_t at the next step
Combine weights	unaffected as a term	*also* unaffected — weights come from the unadjusted scores

$$b_i^{(u+1)} = b_i^{(u)} - \gamma \, \operatorname{sgn}\left(c_i^{(u)} - \bar{c}^{(u)}\right)$$

DeepSeek-V3 reports a load-balancing strategy without a batch-wise auxiliary loss, plus one very small complementary *sequence-wise* term. [REPORTED: DeepSeek-V3 technical report]

"Auxiliary-loss-free" is a claim about an absent objective term. It is not a claim that no balance control exists.

Dispatch and combine

From concept to code: shared plus routed, in one buffer

The layer formula and the route record appear together in the reference implementation's MoE forward pass.

from **DeepSeek-V3/inference/model.py**

```
weights, indices = self.gate(x)
counts = torch.bincount(indices.flatten(), minlength=self.n_routed_experts)
    # ...
    idx, top = torch.where(indices == i)
    y[idx] += expert(x[idx]) * weights[idx, top, None]
z = self.shared_experts(x)
return (y + z).view(shape)
```

From concept to code: shared plus routed, in one buffer

The layer formula and the route record appear together in the reference implementation's MoE forward pass.

from **DeepSeek-V3/inference/model.py**

```
weights, indices = self.gate(x)
counts = torch.bincount(indices.flatten(), minlength=self.n_routed_experts)
    # ...
    idx, top = torch.where(indices == i)
    y[idx] += expert(x[idx]) * weights[idx, top, None]
z = self.shared_experts(x)
return (y + z).view(shape)
```

`idx` is the combine destination, `top` is the selection rank q, `weights[idx, top]` is $w_{i,e}$, and `counts` is the load vector c. `view(shape)` is the type contract. [INSPECTED: DeepSeek-V3 `inference/model.py`]

The combine invariant, priced on one coordinate

Slot	Routed accumulation	Shared	Final
p_1	$(14/17) \cdot 17 + (3/17) \cdot 34 = 20$	1	$y_1 = 21$

[DERIVED: chapter's four-token no-drop trace]

The combine invariant, priced on one coordinate

Slot	Routed accumulation	Shared	Final
p_1	$(14/17) \cdot 17 + (3/17) \cdot 34 = 20$	1	$y_1 = 21$
p_2	$(5/7) \cdot 7 + (2/7) \cdot 14 = 9$	2	$y_2 = 11$

[DERIVED: chapter's four-token no-drop trace]

The combine invariant, priced on one coordinate

Slot	Routed accumulation	Shared	Final
p_1	$(14/17) \cdot 17 + (3/17) \cdot 34 = 20$	1	$y_1 = 21$
p_2	$(5/7) \cdot 7 + (2/7) \cdot 14 = 9$	2	$y_2 = 11$
p_3	$(9/16) \cdot 16 + (7/16) \cdot 32 = 23$	3	$y_3 = 26$

[DERIVED: chapter's four-token no-drop trace]

The combine invariant, priced on one coordinate

Slot	Routed accumulation	Shared	Final
p_1	$(14/17) \cdot 17 + (3/17) \cdot 34 = 20$	1	$y_1 = 21$
p_2	$(5/7) \cdot 7 + (2/7) \cdot 14 = 9$	2	$y_2 = 11$
p_3	$(9/16) \cdot 16 + (7/16) \cdot 32 = 23$	3	$y_3 = 26$
p_4	$(4/7) \cdot 7 + (3/7) \cdot 14 = 10$	4	$y_4 = 14$

[DERIVED: chapter's four-token no-drop trace]

The combine invariant, priced on one coordinate

Slot	Routed accumulation	Shared	Final
p_1	$(14/17) \cdot 17 + (3/17) \cdot 34 = 20$	1	$y_1 = 21$
p_2	$(5/7) \cdot 7 + (2/7) \cdot 14 = 9$	2	$y_2 = 11$
p_3	$(9/16) \cdot 16 + (7/16) \cdot 32 = 23$	3	$y_3 = 26$
p_4	$(4/7) \cdot 7 + (3/7) \cdot 14 = 10$	4	$y_4 = 14$

[DERIVED: chapter's four-token no-drop trace]

Invariant (Dispatch and combine correctness)
Every non-dropped route entry (i, e) contributes exactly once to token i's output, with its own route weight. The final tensor preserves the original token order, hidden width, and declared dtype assumptions.

The combine invariant, priced on one coordinate

Slot	Routed accumulation	Shared	Final
p_1	$(14/17) \cdot 17 + (3/17) \cdot 34 = 20$	1	$y_1 = 21$
p_2	$(5/7) \cdot 7 + (2/7) \cdot 14 = 9$	2	$y_2 = 11$
p_3	$(9/16) \cdot 16 + (7/16) \cdot 32 = 23$	3	$y_3 = 26$
p_4	$(4/7) \cdot 7 + (3/7) \cdot 14 = 10$	4	$y_4 = 14$

[DERIVED: chapter's four-token no-drop trace]

Invariant (Dispatch and combine correctness)
Every non-dropped route entry (i, e) contributes exactly once to token i's output, with its own route weight. The final tensor preserves the original token order, hidden width, and declared dtype assumptions.

Expert buffers are an execution layout. $Y = (21, 11, 26, 14)^\top$ in token order is the contract.

Failure modes and what comes next

Three failure surfaces, three different diagnostics

Symptom	Minimal evidence	What it does *not* prove
Hot or idle expert	per-expert route counts, declared capacity policy	that a rank or network link is the bottleneck

Three failure surfaces, three different diagnostics

Symptom	Minimal evidence	What it does *not* prove
Hot or idle expert	per-expert route counts, declared capacity policy	that a rank or network link is the bottleneck
Wrong-slot combine	route id, expert id, weight, dispatch and combine slots, token-by-token reference output	anything, until compared against a reference

Three failure surfaces, three different diagnostics

Symptom	Minimal evidence	What it does *not* prove
Hot or idle expert	per-expert route counts, declared capacity policy	that a rank or network link is the bottleneck
Wrong-slot combine	route id, expert id, weight, dispatch and combine slots, token-by-token reference output	anything, until compared against a reference
Capacity overflow	declared C_e, retained and overflowed routes, named policy	whether the output changed or the work was merely deferred

Three failure surfaces, three different diagnostics

Symptom	Minimal evidence	What it does *not* prove
Hot or idle expert	per-expert route counts, declared capacity policy	that a rank or network link is the bottleneck
Wrong-slot combine	route id, expert id, weight, dispatch and combine slots, token-by-token reference output	anything, until compared against a reference
Capacity overflow	declared C_e, retained and overflowed routes, named policy	whether the output changed or the work was merely deferred
Near-tie drift	pre-selection scores, dtype, tolerance, tie-break rule	that low-precision routes must match exactly

Three failure surfaces, three different diagnostics

Symptom	Minimal evidence	What it does *not* prove
Hot or idle expert	per-expert route counts, declared capacity policy	that a rank or network link is the bottleneck
Wrong-slot combine	route id, expert id, weight, dispatch and combine slots, token-by-token reference output	anything, until compared against a reference
Capacity overflow	declared C_e, retained and overflowed routes, named policy	whether the output changed or the work was merely deferred
Near-tie drift	pre-selection scores, dtype, tolerance, tie-break rule	that low-precision routes must match exactly

A shape-correct tensor can still carry a corrupt route. That is why metadata failures are more dangerous than crashes: the dimensions all look right.

Concept-to-code map for the chapter

- **router scores, top-k indices, route weights** $\rightarrow$ `DeepSeek-V3/inference/model.py` *(Gate)*

Concept-to-code map for the chapter

- **router scores, top-k indices, route weights** $\rightarrow$ `DeepSeek-V3/inference/model.py` *(Gate)*
- **shared plus routed execution, load counts** $\rightarrow$ `DeepSeek-V3/inference/model.py` *(MoE)*

Concept-to-code map for the chapter

- **router scores, top-k indices, route weights** → `DeepSeek-V3/inference/model.py` *(Gate)*
- **shared plus routed execution, load counts** → `DeepSeek-V3/inference/model.py` *(MoE)*
- **expert, group, and route-scale fields** → `DeepSeek-V3/inference/configs/config_671B.json`

Concept-to-code map for the chapter

- **router scores, top-k indices, route weights** → `DeepSeek-V3/inference/model.py` *(Gate)*
- **shared plus routed execution, load counts** → `DeepSeek-V3/inference/model.py` *(MoE)*
- **expert, group, and route-scale fields** → `DeepSeek-V3/inference/configs/config_671B.json`
- **architectural moves and interventions** → `DeepSeek-MoE/README.md`

Concept-to-code map for the chapter

- **router scores, top-k indices, route weights** → `DeepSeek-V3/inference/model.py` *(Gate)*
- **shared plus routed execution, load counts** → `DeepSeek-V3/inference/model.py` *(MoE)*
- **expert, group, and route-scale fields** → `DeepSeek-V3/inference/configs/config_671B.json`
- **architectural moves and interventions** → `DeepSeek-MoE/README.md`
- **dispatch/combine across ranks** → `DeepEP/README.md`

Concept-to-code map for the chapter

- **router scores, top-k indices, route weights** → `DeepSeek-V3/inference/model.py` *(Gate)*
- **shared plus routed execution, load counts** → `DeepSeek-V3/inference/model.py` *(MoE)*
- **expert, group, and route-scale fields** → `DeepSeek-V3/inference/configs/config_671B.json`
- **architectural moves and interventions** → `DeepSeek-MoE/README.md`
- **dispatch/combine across ranks** → `DeepEP/README.md`
- **load histograms as placement input** → `EPLB/README.md`

Concept-to-code map for the chapter

- **router scores, top-k indices, route weights** → `DeepSeek-V3/inference/model.py` *(Gate)*
- **shared plus routed execution, load counts** → `DeepSeek-V3/inference/model.py` *(MoE)*
- **expert, group, and route-scale fields** → `DeepSeek-V3/inference/configs/config_671B.json`
- **architectural moves and interventions** → `DeepSeek-MoE/README.md`
- **dispatch/combine across ranks** → `DeepEP/README.md`
- **load histograms as placement input** → `EPLB/README.md`

Route, weight, pack, compute, restore — and never lose the token.

Next: Chapter 5 puts this layer inside a V3 decoder block. In Volume II, DeepGEMM executes the expert-local batches, DeepEP distributes dispatch and combine, and EPLB turns the load vector into a placement plan.