Prefill/Decode Disaggregation

Deep Dive into SGLang — Chapter 24

Yin Yang

Foundation Books Project

Where this chapter sits

This is the chapter where the KV handoff stops being a scheduler detail and becomes a cross-process correctness boundary.

- Prefill and decode become two roles on two workers, joined by a KV-transfer lane and handoff metadata.
- The split is safe only when both sides describe the *same request prefix* — shape agreement is not enough.
- SGLang tracks roles, transfer state, cache locks, staging buffers, and admission as *one lifecycle spread across processes*.

Goal: read any point in the handoff and say who owns the bytes, whether they are executable yet, and which request pays if they are not.

Roadmap

The split and its hazard

The handoff as a picture

From concept to code

The admission invariant

The split and its hazard

Why split prefill from decode

Long prompts and token-by-token decode stress different resources.

- **Prefill** is throughput-heavy: read many prompt tokens, build KV state, tolerate batching that raises GPU utilization.
- **Decode** is latency-sensitive: each new token depends on the last and must keep a steady cadence for interactive serving.

Disaggregation puts these on different workers so a 16K-token prompt burst cannot stall short interactive decode streams.

But the split creates a new correctness problem: the decode worker must continue *exactly* the request state prefill computed — after receiving it over a wire.

The hazard: bytes arrive before they are history

The central hazard is *not* the byte copy. It is that decode may be ready to continue before the transferred KV pages have become executable request history.

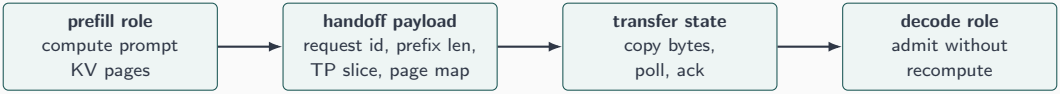
Three moments must stay separate:

1. prefill has **computed** KV rows,
2. transfer has **moved** bytes,
3. decode has **committed** those rows as history for the same request.

A page-shaped buffer can still hold the wrong prefix — another request, another prefix length, or another tensor-parallel slice. The chapter is organized around this lifecycle, not around backend classes.

The handoff as a picture

The handoff crosses four boundaries



Each boundary owns a distinct check:

- who participates (bootstrap + rank mapping),
- which bytes are live (registered memory, staging watermark),
- whether the transfer completed (poll state),
- whether decode may commit (metadata-buffer guard).

Four handoffs, four distinct failure modes.

The compact handoff timing model

Attach a time to the split:

$$T_{\text{handoff}} = T_{\text{prefill}} + T_{\text{transfer}} + T_{\text{decode admission}}.$$

For a 16K-token `Llama-3.1-8B-Instruct` prefill:

$$T_{\text{handoff}} \approx 200\,\text{ms} + 3\,\text{ms} + 1\,\text{ms}.$$

Only the first term is model computation. The other two are serving-system work: bootstrap, allocation, registration, polling, admission.

Why the transfer term cannot be hand-waved — count the bytes.

On-slide derivation: the KV payload is 2 GiB

With 32 layers, $H_{\mathrm{KV}} = 8$ KV heads, head dim $d_h = 128$, two tensors (K, V), two BF16 bytes/element, and 16384 prompt tokens:

$$16384 \cdot 32 \cdot 8 \cdot 128 \cdot 2 \cdot 2 = 2{,}147{,}483{,}648 \text{ bytes} = 2\ \mathrm{GiB}.$$

The formula uses $H_{\mathrm{KV}} \cdot d_h$, *not* model dimension — we send cached key/value heads, not full hidden states.

Bandwidth-only lower bound, same bytes, two transports:

$$\frac{2\,\mathrm{GiB}}{600\,\mathrm{GB/s}} \approx 3.6\,\mathrm{ms} \quad \text{(near NVLink)}, \qquad \frac{2\,\mathrm{GiB}}{50\,\mathrm{GB/s}} \approx 43\,\mathrm{ms} \quad \text{(far RDMA)}.$$

Same prompt, same KV bytes — very different handoff viability by transport.

From concept to code

From concept to code: non-final chunks round to a page

Why chunking keeps early sends "page-clean" lives in one line.

from `python/sglang/srt/disaggregation/prefill.py`

```
def send_kv_chunk(self, req, last_chunk=False, end_idx=None):
    page_size = self.token_to_kv_pool_allocator.page_size
    ...
    if not last_chunk:
        # if not the last chunk and the last page is
        # partial, delay it to the next send
        end_idx = end_idx - end_idx % page_size
```

`end_idx - end_idx % page_size` rounds a non-final chunk down to a page boundary — a partial last page waits for a later send, so only the *final* chunk carries the tail page and auxiliary state.

Concept-to-code map: the disaggregation package

- **Bootstrap connection + rank mapping** → `disaggregation/common/conn.py`
- **Prefill sender, queue, chunk send** → `disaggregation/prefill.py`
- **Decode preallocation, commit, admission** → `disaggregation/decode.py`
- **Decode-side staging ring + watermark** → `disaggregation/common/staging_buffer.py`
- **NIXL registered-memory transfer** → `disaggregation/nixl/conn.py`
- **Mooncake shared transfer engine** → `device_communicators/mooncake_transfer_engine.py`
- **KV event observability stream** → `disaggregation/kv_events.py`

One system spread across files: scheduler-side queues on both roles, and a transfer-backend family that only proves *byte completion* — never decode admission.

The admission invariant

The one invariant everything serves

Invariant (KV-handoff completion)
A disaggregated decode admission is valid only when prefill and decode agree on request identity, prefix length, source/destination page mapping, TP-rank ownership, CP/PP/DP mapping and response counts, transfer completion, optional auxiliary state, and failure/timeout state.

Every other rule — page alignment, staging watermarks, chunking, auxiliary-state commit, the event stream — is a supporting check. None of them creates separate permission to run decode.

What to carry forward

1. Disaggregation changes **process ownership and time**, never the meaning of a request prefix.
2. Keep three moments separate: **computed**, **moved**, **committed**.
3. Bootstrap and rank mapping decide **who participates**; page mapping names **which pages** inside them.
4. A transfer backend proves **byte completion**; decode still **commits** request semantics.
5. The split is worthwhile only when **protected decode cadence** outweighs bootstrap, transfer, commit, and failure-surface cost.

Next: Section 1 states the serving problem, then Sections 2–10 walk the handoff lifecycle from bootstrap to decode admission.