Reasoning and Tool-Call Parsing

Deep Dive into SGLang — Chapter 16

Yin Yang

Foundation Books Project

Where this chapter sits

A stream chunk can end inside a reasoning marker, or halfway through tool-call JSON.

- The model emits *one* token stream; the protocol exposes several channels: visible text, reasoning, tool calls.
- Publish too early and you expose hidden text or finalize the wrong object — and you cannot un-send a streamed delta.
- This is the structured-output sibling: Chapter 15 made tokens *legal*; this chapter decides what the legal text *means*.

Goal: by the end you can look at any streamed byte and say which channel owns it, whether it is safe to emit, and which request state holds it.

Roadmap

The streaming-parser problem

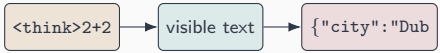
Parser state, emission, and invariants

Boundaries and evidence

The streaming-parser problem

The running example: a Qwen weather stream

`Qwen/Qwen3-0.6B`, OpenAI-compatible: hidden reasoning, then visible text, then a `weather` tool call.



- End a chunk after `{"city":"Dub` — no final tool call yet.
- End a chunk inside `</thi` — cannot expose it as answer text.

The parser's hardest job is knowing when text is not text yet.

The streaming parser as a state machine

Each detokenized chunk x_t drives one step:

$$(p_t,\ e_t,\ h_t) = \mathrm{parse}(p_{t-1},\ x_t).$$

- p_t request-local parser state (reasoning mode, current tool, args).
- e_t the protocol delta safe to publish *now*.
- h_t the held suffix — bytes that may still become a marker or partial JSON.

Correctness means no future token forces retracting an already emitted visible, reasoning, name, or argument byte.

The held suffix, made precise

For marker set M and buffer B, hold the longest suffix that is a *proper prefix* of some marker:

$$k^\star = \max\{k : \ B[\,|B| - k :] \text{ is a proper prefix of some } m \in M\},$$

$$e_t = B[\,0 : |B| - k^\star\,], \qquad h_t = B[\,|B| - k^\star :].$$

- Buffer `<think>2+2</thi`, marker `</think>`: hold `</thi`, emit `<think>2+2`.
- Buffer `<think>2+2t`: nothing starts with `t` — hold nothing.

Hold exactly the ambiguous tail — never ordinary text.

Parser state, emission, and invariants

Two semantic levels meet at the parser boundary

Semantic state

- reasoning text
- visible answer text
- tool name
- tool arguments

Format-specific spelling

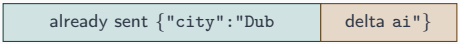
- Qwen `<think>` markers
- Harmony channel tokens
- JSON / XML / Pythonic calls
- parallel-call separators

Keeping these separate is what lets SGLang support many formats without teaching scheduling, sampling, or detokenization any of them.

Stateful diff emission: send each thing once

The OpenAI streaming surface, per response index i:

- visible text → `delta.content`; reasoning → `delta.reasoning_content`; calls → `delta.tool_calls`.
- emit the tool *name* once, then stream only the *new argument suffix*.



Re-emit the whole prefix and you duplicate bytes; emit nothing and you strand the payload.

The one rule: streaming-parser alignment

Invariant (Streaming-parser alignment)
A streaming-parser step is valid only when emitted bytes, held suffixes, request identity, parser mode, parser lifetime, batch row, protocol format, and the tool-execution boundary all describe the same response stream.

The failure is quiet: a partial `</think>` shown as text, a partial JSON published as a complete call, or one response's buffer leaking into another's. Each is a serving-layer state-machine bug, not a crash.

From concept to code: hold only the stable prefix

Harmony's `prefix_hold` is the held-suffix rule, in code.

from `python/sglang/srt/parser/harmony_parser.py`

```
def prefix_hold(text, tokens):
    """Holds back the longest suffix of 'text' that could be a prefix
    of any token. Returns (emit_now, keep_for_later)."""
    max_hold = 0
    for tok in tokens:
        L = min(len(tok) - 1, len(text))
        for k in range(L, 0, -1):
            if tok.startswith(text[-k:]):
                max_hold = max(max_hold, k); break
    if max_hold == 0:
        return text, ""
    return text[:-max_hold], text[-max_hold:]
```

`len(tok) - 1` is "proper" prefix; the returned pair is exactly (e_t, h_t).

Boundaries and evidence

The parser boundary ledger

Boundary	Owner	Parser must *not*
Sampling	logits processors, token selection	change probabilities or KV
Structured output	grammar / structural-tag backend	treat a constraint as a parsed call
Parser emission	reasoning + tool-call detectors	execute the tool
Application	client / serving app code	rewrite streamed parser history

Parsing a call creates protocol output; it does not run the tool and does not prove the arguments are semantically valid.

What the measurements must separate

Parser cost and correctness hide in distinct event fields:

Concept pressure	Direct event field
Held-suffix minimality	held bytes, wait flag, visible bytes
Prefix-stable arguments	argument bytes, duplicate-byte count
Request-local state	parser name, response index, tool index
Format coverage	parser inventory, structural-tag support

Zero duplicate emitted bytes is the headline: the probe shows held suffixes that wait and arguments that stream once — not inferred from end-to-end latency.

What to carry forward

1. Parsing splits **one stream** into protocol channels; it runs after detokenization, before the response is emitted.
2. The step is $(p_t, e_t, h_t) = \text{parse}(p_{t-1}, x_t)$ — emit the stable prefix, **hold** the ambiguous suffix.
3. **Prefix stability**: a streamed delta can never be retracted; the tool name is sent once, arguments stream as diffs.
4. **Streaming-parser alignment** is the one invariant; finish-time flush closes the contract on any held bytes.
5. Every concept has a home in `parser/` and `function_call/`; that link is the spine of this book.

Next: Chapter 17 turns to speculative decoding — proposing many tokens at once and verifying them against the target model.