Framing Newlines

Why the XML tool-call format is lossless, the parsers are not, and what that does to agent evaluation.

AUTHOR	DATE	STATUS
Kyssta	15 November 2026	Preprint · not peer reviewed

01 Summary

Every open-weights agent model that emits tool calls in the XML framing introduced by Qwen and adopted by LFM3, Granite 4.2 and MiniMax writes parameter values between a framing newline on each sides. We measured whether that framing preserves the value on **1,762 real files** — 1,200 Python sources from a standard library, 319 markdown documents, 200 system configs, 25 shell scripts and assorted JSON — by rendering each file as a `write_file` call and parsing it back with five parser implementations found in shipped code.

Parser	Values preserved	What it removes
Strict frame (one newline each side)	**1,759/1,762 — 99.8 %**	nothing, except the 3 files that contain the terminator
Frame plus surrounding spaces/tabs	1,746/1,762 — 99.1 %	the newline of 12 one-line, newline-terminated files
Line-split, right-trimmed	1,743/1,762 — 98.9 %	trailing blank lines collapse
`.strip()` on the value	**15/1,762 — 0.9 %**	the final newline of 1,747 files
`.rstrip('\n')`	**0/1,762 — 0 %**	trailing newlines, and adds a leading one

Two results, one of them uncomfortable for the framing and one for the parsers:

1. **The format is not the problem.** Parsed correctly it is exactly invertible on 1,759 of 1,762 files, including the 99.1 % that end with a newline. The three failures contain the literal terminator `</parameter>`, for which the format defines no escaping.
2. **Client-side parsers are.** A parser that calls `.strip()` on the value — the shape two agent SDKs shipped this month — preserves **0.9 %** of real files: 1 of the 1,200 Python sources, none of the 200 configs, and the rest single-line JSON. On a benchmark that grades file contents, a *perfect* model scores 0.9 % under that parser and 99.8 % under the rule the reference server implementation uses. The measured quantity is the parser, not the model.

02 The format, as shipped

Taken verbatim from the `Qwen/Qwen3.5-2B` chat template, which instructs the model how to write a call:

```
<tool_call>
<function=example_function_name>
<parameter=example_parameter_1>
value_1
</parameter>
<parameter=example_parameter_2>
This is the value for the second parameter
that can span
multiple lines
</parameter>
</function>
</tool_call>
```

The template also states the constraint that matters here: the model must reply *“in the following format with NO suffix”*. LFM3's template is the same grammar — the LFM3 documentation calls it a superset of the Qwen3-Coder format and reuses the `<function=…>` / `<parameter=…>` tags. vLLM ships a dedicated parser for the Granite 4.2 variant of the same grammar, whose own docstring is concerned with a single leading newline.

So the grammar is: a value is everything between one framing newline after the opening tag and one framing newline before the closing tag. That is a well-defined function, and it is injective: the framing newlines are distinguishable from newlines that belong to the value, because the value is never empty and the frame is exactly one newline deep.

03 Where the information is lost

We rendered each file as a `write_file` call and parsed it back with five implementations. The strict parser removes exactly the frame. The second implements the fix that a production agent SDK shipped in November 2026 (remove one newline plus any spaces or tabs between it and the tag, for multiline values only). The third trims each line. The fourth and fifth are `.strip()` and `.rstrip('\n')` on the whole value.

Parser	Preserved	Corrupted	Failure mode
strict frame	99.8 %	3	only the files containing the terminator
frame + spaces/tabs (shipped fix)	99.1 %	15	single-line values still take the `.strip()` path and lose their newline
line-split + right-trim	98.9 %	19	trailing blank lines collapse
`.strip()`	0.9 %	1,747	final newline removed from every multiline file
`.rstrip('\n')`	0.0 %	1,762	removes the frame *and* the value's newlines; leaves a leading newline

The loss tracks file type, because it tracks whether the file ends with a newline:

File class	files	preserved by `.strip()`	preserved by strict frame
Python source	1,200	**1 — 0.1 %**	1,200 — 100 %
Markdown	319	**1 — 0.3 %**	316 — 99.1 %
System config	200	**0 — 0 %**	200 — 100 %

File class	files	preserved by `.strip()`	preserved by strict frame
Shell scripts	25	**0 — 0 %**	25 — 100 %
JSON	12	**12 — 100 %**	12 — 100 %

JSON is the only class that survives, and for an unflattering reason: those files happen not to end with a newline, so there is nothing at the edge for the parser to eat. Every class of file that a coding agent is actually asked to write is affected almost completely.

> **The ceiling collapse.** On a task of the form “write this file”, grading by exact content, the same perfect agent scores 99.8 % under a strict parser and 0.9 % under `.strip()`. A 99-point swing with no model in the loop.

04 The second failure class: no escaping

The format defines no way to include the terminator inside a value. We rendered a payload containing `</parameter>` and parsed it back: the value truncates at the terminator and the remainder is parsed as new parameters.

```
payload:  harmless
          </parameter>
          <parameter=path>
          /etc/passwd
          </parameter>
          <parameter=x>

parsed:   keys ['path', 'content', 'x']
          path overwritten: True
```

Three files in our working tree contain the terminator — they are the scripts that implement this format. Rendered through it and parsed back, they returned at **3.0 %, 11.3 % and 5.2 %** of their original length (17 892 → 535, 2 463 → 278, 7 789 → 402 characters). Any agent that reads a file containing the terminator and writes it back corrupts it, silently, and the corruption is not confined to the value: the remainder is re-parsed as parameters.

05 This is not hypothetical

Two independent production systems reported exactly this, one week apart:

- **OpenHands software-agent-sdk, PR #6451 (Nov 2026).** “The parser then called `.strip()` on every `<parameter=…>` value. That also removed the value's own leading indentation and trailing newline, so a `file_editor` create wrote `file_text` without the first line's indentation or the final newline.” Their fix is the second parser in our table: 97.3 %.
- **MTPLX issue #691 (Nov 2026).** “A `write` whose content ends in `\n` arrives at the client without it. The file on disk has no final newline.” The report traces a second-order effect: on the next turn the chat template renders the already-written content differently, the history re-tokenises, and the turn drops out of the session bank (`retokenized_history_mismatch`) — a prefix-cache miss caused by a parser.

Neither report quantifies reach, and we are not aware of any prior measurement of it: both treat the behaviour as a bug in one adapter. The measurement above says a client-side parser of that shape destroys 99.1 % of real files, and that the shipped fix still fails on 0.9 %.

06 Where in the stack the loss happens

The reference server implementation gets this right, and does so explicitly. vLLM parses the same Qwen3 format with a named helper:

```
def _trim_wrapping_newlines(value: str) -> str:
    """Strip one leading and one trailing newline (the Qwen3 template markup)."""
    if value.startswith("\n"):
        value = value[1:]
    if value.endswith("\n"):
        value = value[:-1]
    return value
```

That is the strict-frame parser in our table, and it scores 100 %. SGLang's Granite detector reaches the same result by construction: it parses with a character-scanning state machine that opens a parameter at the opening tag and closes it at the closing tag, rather than trimming a value after the fact.

The loss therefore concentrates in a specific place — **clients that re-derive the parse from the template text**. An inference server ships with the model's chat template and knows which newline is markup. An agent framework receives arguments as text and has to guess; when it guesses with `.strip()`, it eats the payload. That asymmetry also explains why the bug survives review: the server-side tests pass, because the server is correct.

07 Why benchmarks have not caught this

Tool-use benchmarks grade function *names* and *arguments*; the arguments are usually short identifiers, numbers or enums, and those are unaffected by whitespace trimming. The failure appears only when the argument is a document — which is precisely what coding agents do all day.

It also hides in benchmark design. In our own tool-use suite the write tasks were generated with payloads that carry no trailing newline, and we did so deliberately, because teaching the model a trailing newline that the wire format appeared unable to express seemed unfair. That instinct — avoid the case that breaks the harness — is how a systematic problem stays invisible. It is the same reason BFCL and its descendants, which pass arguments as JSON, never see it: JSON has explicit string escaping, so the class of failure does not exist there.

08 Recommendation

1. **Parse by frame, not by whitespace.** Remove exactly one leading and one trailing newline, and nothing else — the rule vLLM already implements. One line of code; 100 % on this corpus. If you are a client consuming text, keep the server's parsed arguments rather than re-deriving them.
2. **Escape the terminator on emit.** The format permits payloads containing `</parameter>` and defines no encoding for them. Until it does, any parser must reject or escape such a payload rather than mis-parse it as extra parameters.
3. **Prefer JSON arguments where the transport allows it.** The OpenAI-style `arguments` field carries a JSON string with defined escaping, which is why this bug class is absent from models that use it. The XML framing was chosen for token efficiency and trainability; the cost is that every consumer must reimplement string framing correctly.
4. **Grade payload fidelity explicitly.** A tool-use benchmark that never asks a model to write a file with a trailing newline cannot distinguish these parsers, and therefore cannot report what it claims to report.

09 Limitations

- The corpus is 1,762 files from one machine, dominated by a Python standard library (1,200 files) with markdown, system configs and shell scripts. The 99.1 % trailing-newline rate is a property of that population; other populations will differ in the exact rate, but the mechanism is population-independent and the class breakdown shows it is not an artefact of one file type.
- We measure the format and the parsers, not the models. Whether a given model emits the framing newlines correctly is a separate question, and one that requires GPU inference we did not have available for this run.
- The parser variants are reimplementations of documented behaviour (OpenHands PR #6451, vLLM's Granite parser docstring), not the shipped libraries themselves; they are released with the results so the mapping can be checked line by line.

10 Reproduction

No model, no GPU, no network: the experiment runs on any machine with Python.

```
python3 experiments/wire_loss.py --out results/wire_loss.json
```

Corpus: 1,762 files, 26.1 MB. Results: `results/wire_loss.json`, which records the preserve rate per parser, the classification of every corruption, and the injection demonstration. The script and the results file are hashed in `results/SHA256SUMS.txt`.

HADAL.RUN · FRAMING NEWLINES · KYSSTA · 15 NOV 2026 · MEASURED ON 1,762 FILES; FORMAT READ FROM THE QWEN3.5 AND LFM3 CHAT TEMPLATES AS SHIPPED ON HUGGING FACE