

Chat Metrics Manual — Interpreting Performance & Quality Signals

This guide explains every metric surfaced by the **Chat • Web Search + Memory with Metrics** page and how to interpret them when running your bot in **General**, **RAG**, or **Web Search** modes. It's written for a semi-technical audience—no ML background required.

1) Where metrics come from

Your server (`rag_mem_server.js`) returns a rich response that includes a **metrics** block, plus helpful context like **sources**, **mode**, **memory** hits, and a **running summary**. The front end reads those fields and displays them alongside each assistant reply.

At a high level: - **Server measures**: end-to-end request time on the server, time inside the LLM call(s), summary generation time, token usage. - **Client measures**: wall-clock time the browser spent waiting for the response (shown in Perf Runner table as *Client ms*).

2) Quick map of fields

Section	Field	What it is	Why it matters
Reply	<code>reply</code>	Final assistant text returned by the model.	Sanity check: if blank or very short, investigate caps, tool calls, or errors.
Sources	<code>sources[]</code>	Top citations (RAG docs or Web results).	Trust & traceability; compare which pipeline produced the answer.
Mode	<code>mode</code>	One of <code>general</code> , <code>rag</code> , or <code>web</code> .	Latency and resource use differ by mode; set expectations.
Memory	<code>memoryUsed</code> / <code>memoryTop[]</code>	Count and top matches from conversation memory.	Confirms continuity; too many/too old hits can distract.
Summary	<code>runningSummary</code>	Rolling bullet summary of the session.	Useful for long chats and coreference; costs extra latency.
Metrics	<code>metrics{...}</code>	Latency, throughput, and token use (see below).	Core signals for sizing hardware, tuning tokens, and SLAs.

3) Metrics (server-side) — definitions & formulas

3.1 Latency & timing

- `metrics.apiDurationMs`

Definition: Milliseconds spent in the **main LLM completion**.

Why: This isolates model latency from other work (RAG retrieval, JSON marshalling, etc.). Track the impact of model choice and token sizes here.

- `metrics.summaryApiMs`

Definition: Milliseconds spent generating/updating the **running summary** (best-effort; may be null).

Why: Summaries improve long-context chats but add cost/latency. Toggle this on/off at the server level to understand trade-offs.

- `metrics.totalDurationMs`

Definition: Full server time from request receipt to response JSON.

Why: Closest to what users feel (minus network). Includes retrieval, search, memory lookups, the main LLM call, and optional summary.

Typical relationships: `totalDurationMs ≥ apiDurationMs`. If `totalDurationMs` is much larger, your bottleneck may be web search latency, file I/O, or embeddings.

3.2 Token usage & throughput

- `metrics.usage` (if provided by the model)
- `prompt_tokens` : tokens in the prompt (system, user, RAG context, memory).
- `completion_tokens` : tokens generated by the model.
- `total_tokens = prompt_tokens + completion_tokens`.

Why: Tokens directly affect cost and latency. Large prompt tokens usually dominate; reduce context size or chunk count if needed.

- `metrics.tokensPerSec`

Definition: `total_tokens / (apiDurationMs / 1000)` when usage is available.

Why: A normalized throughput measure to compare models/hardware even as token sizes vary.

- `metrics.charsPerSec`

Definition: `reply.length / (apiDurationMs / 1000)`; a fallback throughput proxy when token usage is missing.

Why: Quick sense of output speed; less precise than tokens but universally available.

3.3 Model identifier

- `metrics.model`

Definition: The model that actually produced the reply (may reflect server fallback).

Why: Crucial for A/B tests and when diagnosing changes in latency or output quality.

4) Context fields — why they're logged

- `sources` (RAG or Web Search)
Why: Improves trust and debuggability. If answers look off, check the top sources—maybe the wrong doc was retrieved or the search query was ambiguous.
 - `mode`
Why: Expected latency differs:
 - `general` → fastest (no retrieval/search)
 - `rag` → adds embedding similarity search + longer prompt
 - `web` → adds outbound HTTP calls and longer prompts
 - `memoryUsed` / `memoryTop[]`
Why: Verifies if conversation context is being reused. Low/zero hits across turns may mean your memory threshold is too strict or the `sessionId` changed.
 - `runningSummary`
Why: Helps the assistant keep track of goals, constraints, and entities in long chats. If the summary lags behind, consider turning it off for latency tests.
-

5) Interpreting the numbers — practical guidance

5.1 Establish baselines

1. **General mode** with a short prompt (e.g., “Say hi”).
Expect very low `apiDurationMs` and `totalDurationMs`. Use this to benchmark your network+server overhead.
2. **RAG mode** on a known doc.
Expect `totalDurationMs` to increase due to retrieval and longer prompts. If increases are extreme, tune `topK`, `minScore`, or chunk sizes.
3. **Web mode** with a stable query.
Expect higher `totalDurationMs` due to outbound HTTP latency; measure variance across time of day and provider (SerpAPI vs. Bing).

5.2 Red flags & what to try

- `reply` **empty or very short**
- Check token caps (server env `MAX_COMPLETION_TOKENS`, request `maxCompletionTokens`).
- Ensure tool-only responses are disabled if your model tends to prefer tools:
`DISABLE_TOOL_CALLS=1` (server) for testing.

- Watch server log for messages like “Empty reply from model... finish_reason: length”. Raise caps or shorten prompt.
- `totalDurationMs` $\gg$ `apiDurationMs`
- Likely I/O bound. Profile web search time, indexing, or disk reads. Consider caching search results and embeddings.
- **Huge** `prompt_tokens`
- Trim RAG context (`topK`), lower chunk size, or introduce a ranker that selects fewer/better passages.
- Avoid dumping entire summaries or long prior turns; summarize aggressively.
- **Low** `tokensPerSec` **vs expectations**
- Check if temperature or model changed. Some models stream/generate slower.
- Ensure your token cap isn’t throttling output mid-sentence.

5.3 What “good” looks like (rules of thumb)

These are broad ballparks; real numbers depend on model/hardware.

- **General:** `apiDurationMs` < 600 ms for short prompts; `totalDurationMs` close to `apiDurationMs`.
- **RAG:** Add 100–400 ms for retrieval (in-memory). Prompt tokens may jump x2–x10.
- **Web:** Add 300–1500 ms for search (network). High variance is normal; use caching in production.

6) Perf Runner — batch testing

The Perf Runner calls your `/perf` endpoint with a configurable **count**, **concurrency**, and **delay**. It reports per-run client wall time and the aggregated summary.

Use it to: - Validate that higher concurrency doesn’t starve the GPU/CPU.

- Compare models with the same prompt and caps.

- Spot outliers: track p50/p90/p99 latencies across runs.

Tip: keep `temperature=0-0.2` during perf tests so variability comes mainly from infra, not sampling.

7) Tuning checklist (latency vs. quality)

1. Cap and compress

2. Reduce `topK` or prefer shorter chunks (sentence-aware chunking already helps).
3. Summarize previous turns into a short running summary; don't paste full history.

4. Throttle optional work

5. Disable summary on hot paths if not needed for that flow.
6. Cache search results briefly to avoid duplicate HTTP calls.

7. Prevent tool-only stalls

8. For some models, set `DISABLE_TOOL_CALLS=1` during perf to force pure text outputs.

9. Right-size token caps

10. Start with server env `MAX_COMPLETION_TOKENS` (e.g., 1500–3000) and override per request for long answers.
11. If you see `finish_reason: length`, increase the cap or reduce prompt size.

12. Concurrency

13. Increase worker count gradually while watching p95 latency and error rates.
14. Avoid saturating your CPU with embeddings; batch (your server already batches at 64).

8) Troubleshooting FAQ

Q: Why do I see references but no text?

A: The model likely produced tool calls or hit a token cap. For testing, set `DISABLE_TOOL_CALLS=1` and raise `MAX_COMPLETION_TOKENS`. Also verify the server logs—empty replies are logged with the first choice metadata.

Q: Tokens aren't showing in `metrics.usage`.

A: Some model families omit usage. Fall back to `charsPerSec` and keep `reply.length` as a crude size signal.

Q: Web search feels slow.

A: Measure the external call time and enable caching. You can also reduce the number of results included in the prompt.

Q: Memory hits seem irrelevant.

A: Lower `memTopK` or raise the threshold. Check timestamps in `memoryTop`—older hits may be decayed less in your current config.

9) Field reference (copy/paste)

```
{
  "reply": "string",
  "sources": [{ "title": "string", "url": "string", "snippet": "string" }],
  "mode": "general|rag|web",
  "sessionId": "string",
  "memoryUsed": 0,
  "memoryTop": [{ "id": "string", "ts": 0, "score": 0.0 }],
  "runningSummary": "string",
  "metrics": {
    "model": "string",
    "apiDurationMs": 0,
    "summaryApiMs": 0,
    "totalDurationMs": 0,
    "usage": { "prompt_tokens": 0, "completion_tokens": 0, "total_tokens": 0 },
    "chars": 0,
    "tokensPerSec": 0.0,
    "charsPerSec": 0.0
  }
}
```

10) Suggested thresholds to watch (starting points)

- **p95** `totalDurationMs`
 - General: < 1500 ms
 - RAG: < 2500 ms
 - Web: < 4000 ms (without caching)
- `tokensPerSec`
 - Keep an eye on drops >30% between builds or model versions.
- `prompt_tokens`
 - Alert if >8–12k persistently (risk of context overflow and high cost).

Adapt these to your hardware and SLAs.

Final tip

For clean apples-to-apples performance comparisons, lock down: model, temperature, token caps, and prompt length. Then vary one thing at a time (e.g., RAG `topK` or web search provider). Record `apiDurationMs`, `totalDurationMs`, and tokens—these three together explain most latency and cost swings.