Title: Router Slot/Parameter Extraction Benchmark (Local LLMs via Ollama)
Author: Alperen TEKİN alperen.tekin@intellica.net
Consultant: Selcan Yükçü selcan.yukcu@pia-team.com
Date: 17 Dec 2025
Environment: Local machine, Ollama, Python benchmark scripts

1. Executive Summary

This report evaluates multiple local LLMs for a router-style **parameter extraction** task. The goal is to reliably extract one or more structured parameters from user answers and return **strict JSON** suitable for downstream automation.

We progressively increased task difficulty across several benchmark groups:

1. single-step extraction (one question/answer),
2. multi-turn transcript (5 Q/A pairs),
3. opportunistic multi-field extraction when the user provides extra information,
4. mixed “unified” scenarios combining both behaviors.

We compare models primarily on:

- **Accuracy** (per-test success and per-field update accuracy),
- **Robustness** (JSON parse stability, forbidden behaviors),
- **Latency** (average time per test / TTFT).

Disclaimer

All of these tests below ran on CPU, considering the environment will not have appropriate GPUs to run superior versions of these models — no vLLM.

2. Problem Definition

2.1 Task Overview

We simulate a router that asks the user for missing parameters required to configure jobs such as:

- **read_sql**
- **write_data**
- **email**

The LLM receives:

- the **last router question**,
- the **user answer**,
- the **current parameter state (Existing Params)**,

- optionally a **Missing params** list or multi-turn transcript.

The LLM must return JSON only:

```
{"params": {...}}
```

2.2 Core Requirements

- **Strict JSON only** output.
- **Never delete keys** from ExistingParams.
- **Do not overwrite existing non-empty values** (unless the benchmark allows it explicitly).
- Extract values using defined rules for:
    - identifiers (name/table),
    - booleans (execute_query/write_count/etc.),
    - job-specific allowed keys.

3. Benchmark Design

3.1 Benchmark Groups (Experiments)

Experiment A — Single-step extraction (baseline)

Input format: Last question + User answer + Missing Param + Existing Params
Objective: Extract only the parameter implied by the last question (or leave unchanged if uncertain).
Typical examples:

- Ask for job name → extract `name`
- Yes/no question → map to boolean

Metrics recorded: success rate, parse errors, overwrite bugs, avg latency.

Experiment B — Single-turn multiple parameters (5 Q/A pairs)

Input format: A short transcript: `Q1/A1 ... Q5/A5` + slot order/ExistingParams
Objective: Fill multiple slots in sequence (single call), using answers only.
This tests whether the model stays consistent across multiple Q/A pairs and respects ordering/constraints.

Experiment C — Opportunistic multi-field extraction

Motivation: In real usage, the user may answer with more information than asked.
Example: router asks for email recipient, but user also includes subject/text/cc.

Objective:

- Always satisfy the asked field,
- Additionally fill *other missing fields* **only when confidently extractable** and **only if those fields are currently empty**.

We also applied grouping logic:

- read_sql parameters tend to appear together early,
- then write_data parameters,
- then email parameters.

Experiment D — Unified mixed scenarios (final)
A combined benchmark where each test may be either:

- single-field answer (classic router behavior), or
- multi-field answer (opportunistic fill).

This final set represents the most realistic environment.

Notes applied from system behavior constraints:

- If `execute_query=true`, router later asks schema/table via UI dropdown, so opportunistic extraction for those cases is not necessary.
- We mainly evaluate `execute_query=false` paths and boolean correctness, plus write_data/email opportunistic fills.

4. Models Evaluated

4.1 Candidate Models

- **Qwen3 1.7B** (baseline, fast)
- **Mistral 7B** (tested: faster than 1.7B, accuracy improvement vs 1.7B)
- **Llama2 7B** (planned / tested)
- **Falcon 7B (attempted)** (rejected; very poor latency & accuracy)

4.2 Why These Models

We prioritize:

- local deployment feasibility,
- runtime speed faster than *Qwen3 8B*,
- improved or comparable accuracy to *Qwen3 1.7B*.

5. Evaluation Methodology

5.1 Metrics

Primary metrics

- **Test success rate**: % tests that pass all rules and expected updates.
- **Field-level accuracy**: correct_updates / expected_updates (e.g., 109/132).
- **Parse error rate**: invalid JSON / empty output / truncated output.
- **Bug rates**
  - overwrite bug (changed an existing value),
  - invalid key bug (param not allowed for the job type),
  - unexpected update (filled something that shouldn't be changed).

Performance metrics

- **TTFT (time to first token)**
- **Total time per test**
- Aggregate averages and worst-case observations.

5.2 Common Failure Modes Tracked

- JSON truncation / unterminated strings
- Boolean confusion (mapping answer to wrong boolean field)
- Wrong field selection (e.g., answering `write_count` when asked about `execute_query`)
- Mis-handling dotted identifiers (schema.table splitting)
- Returning ExistingParams (Current) unchanged when extraction is obvious (over-conservatism)

6. Results Summary

6.1 Experiment A — Single-step Extraction

- Model: Qwen3-1.7B (exclusively)
- Total tests: 136
- Success: 134 / 136 → ~%98
- Field accuracy: 134 / 136 → ~%98
- Parse errors: 0
- Avg total time/test: 13.65 sec

Observations:

- Model (Qwen3-1.7B) can effectively extract parameters when it knows there are only one parameter to extract and only one answer is given by the user. Note that the provided system prompt is cruical for boolean expressions' extraction.

6.2 Experiment B — 5-turn Transcript

- Model: Qwen3-1.7B (exclusively)
- Total scenarios: 150
- Scenario success *(5 out of 5)* : 96 / 150 → %64
- Field accuracy: 650 / 750 → %86.67
- Parse errors: 10 / 150 → %6.67
- Avg total time/test: 19.17 sec
- Notes on stability across multiple Q/A pairs:
    - Model was informed about the parameters order via its system prompt.

6.3 Experiment C — Opportunistic Multi-field Extraction

- Model: Qwen3-1.7B (exclusively)
- Total tests: 12
- Success: 12 / 12 → %100
- Field accuracy: 24 / 24
- Avg total time/test: 21.11 sec
- Key wins:
    - email: extraction of `to/subject/text/cc` from a single rich user answer
    - write_data: filling table/schema/connection when user provides them together

Observed limitations:

- dotted identifiers (schema.table) inconsistent handling
- occasional confusion between schema vs connection tokens

6.4 Experiment D — Unified Mixed Scenarios (Final)

Model	*Qwen3-8b*	*Qwen3-1.7b*	*Mistral 7B*	*LLama 2 7B*
Total tests	*72*	*72*	*72*	*72*
Success	*64* ***(%88.8)***	*54* ***(%75)***	*65* ***(%90.2)***	*36* ***(%50)***
Total fields	*132*	*132*	*132*	*132*
Field accuracy	*115* ***(%87.1)***	*109* ***(%82.6)***	*124* ***(%93.9)***	*75* ***(%56.8)***
Parse err / Total err	*8 / 8* ***(%100)****	*4 / 18* ***(%22.2)***	*0*	*2**
Avg time (sec)	*75.865*	*17.132*	*7.575*	*33.702*

*LLama 7B had no parse errors however, had 1 overwrite bug and 1 unexpected-update bug.

*Qwen3-8B's mistake were all on parse errors. These may include; empty response from the model as well as inappropriate parsing functions failures. Observations reveals, first case is more likely. But please read the "Note" below.

Note: Please note that all of these tests were performed in the same environment and with identical parameters; therefore, your results may vary. The primary objective was to evaluate these models under strict hardware constraints—specifically using an Intel® Core™ i7-1165G7 CPU—and to compare them on a proportional basis in order to explore potential replacement options.

Takeaways:

- Unified prompt achieved strong generalization across both single-field and multi-field cases.
- Remaining errors are concentrated in:
    - parse stability (truncation),
    - ambiguous phrasing for booleans,
    - dotted identifier edge cases.

7. Model Comparison (High Level)

Qwen3 1.7B

- Pros: Stable, accurate
- Cons: Very slow, needs specific parsing-focussed prompt optimization.

Qwen3 1.7B

- Pros: fast, stable baseline
- Cons: weaker opportunistic multi-fill; occasional wrong field mapping, requires elaborate prompt engineering

Mistral 7B

- Pros: faster and more accurate than qwen3-1.7b
- Cons: longer TTFT, however still beats qwen3-1.7b in overall response time.

Llama2 7B

- Cons: Slow, not as accurate as qwen3-1.7b. Likely because of un-optimized prompts for this specific model.

Falcon

- Outcome: rejected
- Reasons: extreme latency and near-zero accuracy on early tests

8. Discussion

8.1 What Worked Well

- Grouping parameters by job type (read_sql → write_data → email) reduced invalid-key mistakes.
- Opportunistic fill improved real-world UX (fewer turns) especially for email cases.
- Maintaining strict "no overwrite" rules prevented destructive updates.

8.2 What Still Breaks

- **Parse errors** (truncation/empty output) dominate "unfair" failures despite correct intent.
- **Ambiguous negation / contrast** ("no saving; yes track row count") can flip booleans.
- **Dotted identifiers** remain inconsistent unless explicitly handled.

8.3 Practical Implications

- For production routing, parsing stability may matter as much as raw extraction accuracy.
- Having a minimal fallback (retry once, or enforce JSON via wrapper) can significantly reduce parse-related failures.

9. Observations — Prompt Optimization & Debugging

During early iterations of Experiment A, a small number of failures were traced not to model capability but to **prompt-level ambiguity and output formatting issues**. These were resolved through iterative system prompt optimization.

The prompt optimization process followed a structured approach:

9.1. Explicit Role Definition

The system prompt was refined to clearly position the model as a **parameter extraction component**, not a conversational assistant.
The model was instructed to:

- Extract only information that is **explicitly present** in the user answer
- Avoid reasoning, explanation, or natural language output
- Always respond in a **strict JSON format** under a predefined `params` object

This significantly reduced hallucinated fields and parse errors.

9.2. Boolean Normalization Strategy

Boolean fields (`execute_query`, `write_count`, `drop_or_truncate`) were a major early source of inconsistency.

The prompt was updated to:

- Map **natural language variations** (“no saving”, “preview only”, “don’t track”) to canonical boolean values
- Explicitly disallow `null`, empty strings, or mixed signals for booleans
- Prefer **explicit false** when the user clearly negates an action

This eliminated most false positives and inconsistent boolean outputs.

9.3. “Extract What You See” Principle

The prompt emphasized a strict extraction rule:

Only extract parameters that are directly stated in the user answer.

This avoided:

- Inferring downstream parameters
- Guessing values from context
- Filling fields that are not linguistically grounded in the input

As a result, the model became deterministic and predictable across single-step tests.

9.4. Controlled Opportunistic Fill

While inference was discouraged, the prompt allowed **opportunistic extraction** when:

- Multiple parameters were **explicitly mentioned in a single user response**
- Each extracted value could be clearly mapped to a known slot

This was especially relevant for cases like:

- Email jobs (`to`, `subject`, `text`)
- Write operations where table, schema, and connection were mentioned together

Crucially, opportunistic fill was **limited to explicitly stated information**, not UI-driven or implicit values.

9.5. Output Stability & Parsing Safety

The system prompt enforced:

- No trailing text
- No comments
- No partial JSON responses

This removed early parse errors caused by truncated or mixed outputs.
After prompt stabilization, **parse errors dropped to zero**.

Summary

The final prompt did not “teach” the model new capabilities, but **constrained and clarified** its behavior.
Through role isolation, boolean normalization, strict extraction rules, and controlled opportunistic fill, Qwen3-1.7B achieved near-perfect performance in single-step extraction tasks with stable output and low latency.

10. Conclusion

All in all, the strictness of the router outline plays a critical role in determining model success. When parameter fields are predefined (for example, write_count, job_name, and table_rows) the most effective optimization strategy is to explicitly specify potential user responses for each parameter. For further details, please refer to the SYSTEM_PROMPT sections in the provided code files.

Mentioned test results/files can be found at the repository.