

SpendSight: Privacy-Preserving Personal Finance Q&A with Sub-2B LLMs

Sajag Swami Yash Pandey

Independent Researchers

sajagswami@gmail.com

ya.pandey@outlook.com

Abstract

People often speculate about their personal finances: “*Has my spending on groceries increased compared to past year?*”, “*How does my spending on clothes look for the past 6 months?*”. However, answering them with cloud-hosted LLMs transmits sensitive financial data to external servers, exposing users to data leakage and inferential privacy risks. We present **SpendSight**, a study of fully on-device personal finance Q&A tool calling that keeps all data strictly local. We evaluate seven quantized models (six sub-2B and one 3B) across 115 test cases and five analytical tools, comparing single-agent against dual-agent (router-specialist) architectures. Dual-agent decomposition yields the largest time gains for the Qwen3.5 family. Qwen3.5-0.8B achieves 61.7% Correct Ratio (89.1% Task Accuracy) in 3.0 s with 1.2 GB RAM in dual-agent mode. A 3B reference model (G9v3-3B, Q4_K_M) achieves 71.7% Correct Ratio (90.6% Task Accuracy) at 2.3 s in single agent mode, suggesting that larger models with aggressive quantization can bypass decomposition.

1 Introduction

People often wonder about their personal finances. *Has my spending on dining increased since past month? What were some of my largest purchases last month?* These are natural, everyday questions, and modern large language models (LLMs) can answer them by calling structured analytical tools over a user’s expense database (Schick et al., 2023; Patil et al., 2023; Qin et al., 2024).

The problem is *where* this inference happens. Cloud-hosted assistants require users to transmit their financial queries, and often the underlying data, to external servers. A query like “how much did I spend on therapy last month?” is not merely a database lookup; it serves as a disclosure of a health condition, a temporal pattern, and a financial

commitment. Financial records encode sensitive behavioral patterns that risk data leakage and inferential privacy violations when processed remotely (Staab et al., 2024; Yao et al., 2024).

This creates an underexplored problem at the intersection of financial NLP and on-device inference: *can sub-2B quantized models running entirely on consumer hardware reliably call domain-specific financial tools from informal, colloquial natural language?* If so, on-device inference may possess the potential to eliminate cloud dependency entirely, thereby serving as a privacy guarantee for users.

Existing work on financial tool calling, such as FinMCP-Bench (Zhu et al., 2026) and FinTrace (Cao et al., 2026), evaluates cloud-scale models ($\geq 7B$) on multi-step trajectories. General-purpose on-device models like Octopus v2 (Chen and Li, 2024) and Hammer (Lin et al., 2024) show the potential of smaller models but have not been evaluated on domain-specific financial schemas or under consumer-hardware latency constraints. Our study bridges this gap.

We address three research questions (RQ):

- **RQ1:** Can on-device inference with sub-2B models provide practically viable, private personal finance Q&A analytics (i.e., can users obtain fairly accurate personal finance insights without sending query data to cloud providers)?
- **RQ2:** Does decomposing intent classification from parameter extraction (dual-agent) yield measurable accuracy gains over a joint single-agent architecture?
- **RQ3:** What are the primary failure modes across architectures, and how do they differ in the real-world harms they produce for users relying on their outputs?

This paper presents the following contributions:

1. A **study of on-device personal finance Q&A tool calling**: Qwen3.5-0.8B dual achieves 61.7% strict Correct Ratio (89.1% Task Accuracy) at 3.0 s latency and ≈ 1.2 GB RAM, keeping all data strictly local.
2. A **fine-grained error taxonomy** revealing that dual-agent architectures shift errors from silent parameter corruption toward visible omissions (Avižienis et al., 2004), a potentially favorable error profile shift for financial use cases where spurious constraints silently bias downstream calculations (Li et al., 2023; Levi, 2026).
3. A **comparative evaluation** of single-agent vs. dual-agent across six sub-2B quantized models (0.8B–2.0B): for the Qwen3.5 family, dual-agent reduces latency (up to 48% for Qwen3.5-0.8B) and improves strict correctness (+17.4 pp CR for Qwen3.5-2B); these effects are not uniform across all six model families, as latency in fact increases under dual-agent for EXAONE-4.0-1.2B and the 3B reference model (Table 4).
4. An **extended comparison** with a 3B model (G9v3-3B, Q4_K_M) showing that larger models under aggressive quantization can match/surpass sub-2B dual-agent accuracy in single-agent mode, thereby helping delineate when decomposition might be necessary.
5. A **deployment illustration**: an end-to-end on-device personal finance workflow where a participant tracked snack expenses locally over six months, demonstrating practical feasibility without data leaving the device (presented purely as a qualitative illustration; no causal impact claimed).

2 On-Device Inference as Privacy Protection

Conversational interfaces for personal finance expose a structural issue: routing natural-language queries through cloud LLMs leaks behavioral signals that can be retained, aggregated, and exploited (Staab et al., 2024; Yao et al., 2024). SpendSight addresses this threat by executing all inference *on-device*. By leveraging sub-2B quantized models that run efficiently on consumer hardware (tested on an Apple M1 with 8 GB memory), all user data, queries, and model outputs remain strictly

local, preserving *contextual integrity* (Nissenbaum, 2004).

There are three potential threats users are exposed to when using cloud LLMs: cloud data retention, query-level inference attacks, and third-party data exposure. On-device inference eliminates all three by ensuring queries never leave local infrastructure. While sub-2B models have historically been unreliable for structured tool calling (Kavathekar et al., 2025), our study provides evidence that architectural decomposition (separating intent classification from parameter extraction) may help narrow this reliability gap for certain model families (notably Qwen3.5), making private on-device inference practical for real personal financial Q&A systems.

3 Related Work

Tool-Augmented Language Models. Toolformer (Schick et al., 2023) demonstrated self-supervised tool-use learning; Gorilla (Patil et al., 2023) and ToolLLM (Qin et al., 2024) scaled to thousands of real-world APIs. HuggingGPT (Shen et al., 2023) and ToolAlpaca (Tang et al., 2023) explored multi-step orchestration. APIGen (Liu et al., 2024c) and ToolACE (Liu et al., 2024a) introduced verified pipelines for generating function-calling training data, showing that open-weight models fine-tuned on high-quality synthetic data can match or surpass proprietary cloud models on structured tool invocation. These approaches primarily rely on larger models (≥ 7 B) or dedicated post-training pipelines, leaving off-the-shelf sub-2B on-device inference largely unaddressed.

Function-Calling Evaluation. The Berkeley Function Calling Leaderboard (Patil et al., 2025) established a benchmark using AST substring matching and execution-based validation. Kavathekar et al. (2025) proposed an evaluation suite (FSP, ACS, AVC, Task Accuracy, and Correct Ratio) for small model function calling; we adopt these metrics directly. CallNavi (Song et al., 2025) proposed a two-stage router+specialist approach, similar to our dual-agent design, but evaluates models ≥ 7 B on general-purpose APIs. Jhandi et al. (2025) studied fine-tuning SLMs for agentic tool calling using trajectory-level data, demonstrating that targeted domain adaptation can close the gap with larger models. In contrast, we focus on sub-2B models for domain-specific, single-turn personal finance

tasks without fine-tuning.

On-Device Function Calling. MobileLLM (Liu et al., 2024b), Phi-3 (Abdin et al., 2024), and related surveys (Xu et al., 2024) have explored sub-4B parameter models for mobile deployment. Octopus v2 (Chen and Li, 2024) fine-tuned a 2B model for on-device API calling via functional tokens, achieving competitive latency against GPT-4 on specific tasks. Hammer (Lin et al., 2024) introduced function masking to reduce overfitting to function names, addressing an issue similar to the category normalisation errors we observe. Apple Intelligence (Gunter et al., 2024) validated on-device tool calling at scale with a ~ 3 B model. Kavathekar et al. (2025) found that compliance with producing structured output is the primary bottleneck for SLMs.

Financial NLP and Tool Calling. FinQA (Chen et al., 2021) and FinGPT (Yang et al., 2023) target financial numerical reasoning and generation but focus on text generation rather than structured tool invocation. More recently, FinMCP-Bench (Zhu et al., 2026) evaluates LLM agents for real-world financial tool use under the Model Context Protocol, and FinTrace (Cao et al., 2026) provides trajectory-level evaluation of tool calling for long-horizon financial tasks. Both target cloud-scale models (≥ 7 B) on multi-step trajectories involving multiple financial APIs. Our study is complementary: we focus on *single-turn, on-device* tool calling with sub-2B models for personal finance, a setting where privacy constraints make cloud inference undesirable and where the user directly consumes the output for personal decision-making. Personal finance assistants (Li et al., 2024) require strict privacy, making local execution crucial.

Privacy for Financial Language Models. As per Contextual integrity framework (Nissenbaum, 2004), it is ideal that personal financial data not cross contextual norms. Since LLMs can reconstruct sensitive attributes from text (Staab et al., 2024), on-device processing is critical to eliminate cloud exposure pathways entirely.

4 System Design

4.1 Tool Inventory

SpendSight exposes five analytical tools over a user’s expense database. Table 1 summarises each tool and its parameter space. These tools span core

Tool	ID	Key Params
plot_time_series	1	category, year/month, ranges, months, ignore_rent
plot_distribution	2	category, year/month/day, ranges, months, ignore_rent
plot_comparison_bars	3	category, y1/m1, y2/m2, sm/em/ey, ignore_rent
calculate_total	4	category, year/month/day, remarks, months, ignore_rent
get_top_expenses	5	n, category, year/month/day, months, min_amount, ignore_rent

Table 1: Tool inventory. The five tools span visualisation (1–3), aggregation (4), and retrieval (5).

personal finance use cases that reflect recurring information needs identified in prior work: expenditure totals and category-level breakdowns (Blake and d’Ardenne, 2012; Jäckle et al., 2024), spending trends over time and period comparisons (Hahn et al., 2022), and ranked retrieval (Levi, 2026). They reflect the questions users naturally ask: time series analysis (“*how has my spending changed?*”), categorical breakdown (“*how does the breakdown of my spending look?*”), period comparison (“*how does this month compare to last?*”), aggregation (“*how much did I spend on X?*”), and ranked retrieval (“*what were my biggest purchases?*”).

4.2 Architectural Configurations

We evaluate two architectures for mapping natural language user queries to tool calls.

Single-Agent Pipeline. The single-agent pipeline uses a single prompt with all five tool definitions, parameter schemas, and joint few-shot examples. The model must select the tool and extract parameters in a single pass, outputting a JSON object containing the tool ID (1–5) and arguments for the chosen tool. The prompt includes structured output rules, date-handling instructions, and 73 tool-specific examples aggregated across all 5 tools. Prompt length: $\approx 5,500$ tokens (context window inclusive).

Dual-Agent Pipeline. The task is decomposed into two sequential inference calls:

1. **Router:** A focused intent-classification prompt with 19 few-shot examples and explicit disambiguation rules (e.g., “compare” requires two distinct time periods) elicits a single digit (1–5).
2. **Specialist:** The predicted tool ID selects a tool-specific prompt containing 13–16 exam-

ples for that *one* tool, along with detailed parameter definitions and date-handling rules.

The router sees $\approx 1,000$ tokens; the specialist sees $\approx 1,500$ – $2,000$ tokens with deeper, more directly relevant supervision. This simplifies the task for each model call and provides more relevant in-context examples, though it requires two sequential inference passes. Combined prompt tokens: $\approx 2,700$ per query. Both architectures inject a dynamically generated category vocabulary (11 major categories and ≈ 60 subcategory aliases drawn from the user’s own dataset (Appendix B)) into a `{metadata}` block within the prompt, grounding category parameter extraction in the user’s actual expense taxonomy. These category names are inherently sensitive: they reveal what a user spends money on (e.g., healthcare, dining, donations), making their transmission to third-party cloud providers a privacy risk. In dual-agent mode, only the specialist receives this vocabulary; the router operates on pure intent classification without category context.

4.3 Post-Hoc Parameter Validation

A post-processing validation layer corrects category names deterministically without additional model calls:

1. **Exact match** against the known vocabulary;
2. **Morphological normalisation** (Consider `groceries`→`grocery`, `cafés`→`cafe`)
3. **Edit-distance matching** via `diffib` (cutoff ≥ 0.80)
4. **Token overlap scoring** (Jaccard; cutoff ≥ 0.50).

Primary tables report validated (post-validation) metrics, while the quantitative impact of post-hoc validation across all metrics and models is isolated and illustrated via the validation-delta heatmap (Figure 6).

5 Evaluation Design

5.1 Test Suite

The evaluation comprises 115 test cases across five task groups (Table 2). Queries are written in natural and informal language: abbreviations (“*compare electricity ’24 vs ’25*”), colloquialisms (“*can ya tell me*”), misspellings (“*distribution*”), and

Task Group	Cases	Avg Params	Avg C
Time Series	23	2.9	7.0
Distribution	21	2.5	6.4
Comparison	28	4.4	8.8
Calculate Total	22	2.4	6.3
Top Expenses	21	3.2	6.7
Total	115	3.2	7.1

Table 2: Test case distribution by group. Comparison queries are the most parameter-dense and complex.

domain-specific terms (“*combinis*”, “*nomikais*”, “*shinkansen*”). This mimics how real users interact with personal finance assistants, using casual language with abbreviations and cultural terminology specific to their context.

5.2 Complexity Scoring

Each test case is assigned a difficulty score:

$$C = n_p + d_{\text{date}} + d_{\text{cat}} + d_{\text{rel}} + d_{\text{multi}} + d_{\text{abbr}} + d_{\text{hol}} \quad (1)$$

where n_p = number of expected parameters; $d_{\text{date}} \in \{0-4\}$ scores date specification complexity; $d_{\text{cat}} \in \{0-2\}$ is the category normalisation distance; $d_{\text{rel}} \in \{0, 1\}$ flags relative time expressions; $d_{\text{multi}} \in \{0-2\}$ counts compound parameter groups; $d_{\text{abbr}} \in \{0, 1\}$ flags abbreviated dates; and $d_{\text{hol}} \in \{0, 2\}$ flags calendar knowledge requirements. Scores are binned into three tiers: **L1** ($C \leq 4$, 24 cases), **L2** ($5 \leq C \leq 7$, 45 cases), **L3** ($C \geq 8$, 46 cases). The equal weighting of complexity features in Eq. 1 is a design choice; empirical validation of feature importance is left for future work.

5.3 Evaluation Metrics

We adopt the metric suite of Kavathekar et al. (2025) to assess tool routing and parameter extraction performance:

- **FSP** (Function Selection Performance): Binary indicator of correct tool prediction (i.e. routing).
- **ACS** (Argument Completeness Score): F_1 on argument *name* overlap (ignoring values).
- **AVC** (Argument Value Correctness): Fraction of correctly named arguments whose extracted values also match.
- **Task Accuracy**: F_1 on the combined set $\{\text{tool}\} \cup \{k=v\}$, jointly capturing tool selection and parameter correctness.

Model	Family	Params	RAM
EXAONE-4.0-1.2B	EXAONE-4.0 (2025)	1.2B	2.0 GB
Gemma-3-1B-it	Gemma-3 (2025)	1B	1.6 GB
LFM2.5-1.2B	LFM2.5 (2026)	1.2B	1.4 GB
MiniCPM5-1B	MiniCPM5 (2026)	1B	1.5 GB
Qwen3.5-0.8B	Qwen3.5 (2026)	0.8B	1.3 GB
Qwen3.5-2B	Qwen3.5 (2026)	2B	2.1 GB
G9v3-3B [†]	G9v3 (2026)	3B	2.4 GB

Table 3: Models evaluated. Sub-2B models use Q8_0 GGUF quantization; [†]G9v3-3B uses Q4_K_M. All run via llama.cpp. RAM is average peak RSS.

- **Correct Ratio (CR):** Strict binary indicator: 1.0 only when Task Acc = 1.0, i.e. the entire function call is perfectly correct for execution.

All metrics are averaged across 5 repetitions per test case (temperature = 0.1). 95% bootstrap confidence intervals (Efron, 1979) ($n = 2000$ resamples) are reported in Table 10 (Appendix G), with resampling performed over the 115 test cases.

5.4 Error Taxonomy

Each prediction is classified into one of seven mutually exclusive error types: CORRECT, TOOL_WRONG, PARSE_FAILURE, PARAM_MISSING, PARAM_VALUE_WRONG, PARAM_EXTRA, FORMAT_VIOLATION. This taxonomy is crucial for evaluating the *safety profile* of on-device financial assistants: in personal finance, a silently wrong answer from a spurious parameter might be more harmful than a visibly incomplete one from a missing parameter (Avizienis et al., 2004; Li et al., 2023).

6 Experimental Setup

6.1 Models

All sub-2B models (Table 3) use Q8_0 GGUF quantization; G9v3-3B uses Q4_K_M. All seven models run via llama.cpp (Gerganov, 2023) with full GPU offload on Apple Metal. G9v3-3B was selected as the top-ranked open-weight model under 4B parameters on the Artificial Analysis Intelligence Index at the time of evaluation. Since G9v3-3B supports an optional extended thinking mode, we disabled it (`enable_thinking=false`) for all runs, to keep it comparable to the non-reasoning inference setting used for the other six models.

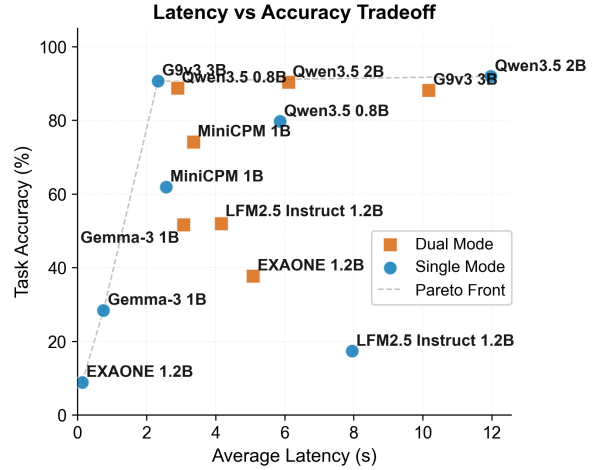


Figure 1: Accuracy-latency trade-off across evaluated models and architectures. Dual-agent Qwen3.5-0.8B (≈ 3.0 s, 89.1% Task Accuracy) sits on the efficient frontier, offering the best accuracy per unit latency. Single-agent Qwen3.5-2B reaches 92.0% Task Accuracy at $4\times$ higher latency (12.0 s).

6.2 Hardware and Inference Protocol

Experiments run on an Apple M1 (8-core CPU, 8-core GPU, 8 GB unified memory). Parameters: temperature = 0.1, min_p = 0.05, n_ctx = 8192, stop tokens [“\n\n”, “ ”]. Each model is warmed up through all six prompt templates (one router + five specialists) before evaluation begins. Each of the 115 test cases is run 5 times per model per architecture, yielding $115 \times 5 \times 7 \times 2 = 8,050$ total observations.

7 Results

7.1 Overall Performance

Table 4 presents the full results. Figure 1 shows the accuracy-latency trade-off across evaluated models and architectures.

Key findings (RQ1, RQ2). (1) Within the Qwen3.5 family, on-device inference can produce practically viable financial tool calls (RQ1): Qwen3.5-0.8B dual achieves 89.1% Task Accuracy (61.7% strict CR) and Qwen3.5-2B dual achieves 90.4% Task Accuracy (77.6% CR). Other sub-2B models show substantially lower performance (37.8%–74.3% dual Task Accuracy), indicating that RQ1 does not hold for sub-2B models as a class. (2) While single-agent Qwen3.5-2B reaches 92.0% Task Accuracy, dual-agent decomposition is preferred for two reasons (RQ2): (a) Strict correctness: Correct Ratio improves from 60.2% to 77.6% (+17.4 pp) for Qwen3.5-2B, representing a marked

Mode	Model	FSP	ACS	AVC	Task Acc	CR	Latency (s)	RAM (GB)
Single	EXAONE-4.0-1.2B	19.3	0.2	0.3	8.9	0.3	0.1	2.0
	Gemma-3-1B-it	27.0	34.0	47.5	30.0	0.9	0.8	1.6
	LFM2.5-1.2B	28.7	9.4	11.7	17.2	3.1	7.8	1.4
	MiniCPM5-1B	71.1	66.4	86.7	62.3	13.0	2.6	1.5
	Qwen3.5-0.8B	95.0	79.6	91.2	79.8	40.2	5.8	1.3
	Qwen3.5-2B	96.5	92.9	97.6	92.0	60.2	12.0	2.1
Dual	G9v3-3B [†]	95.7	91.2	97.0	90.6	71.7	2.3	2.4
	EXAONE-4.0-1.2B	90.6	3.7	3.3	37.8	3.0	5.0	1.9
	Gemma-3-1B-it	55.1	59.9	71.6	51.7	16.5	3.1	1.5
	LFM2.5-1.2B	69.6	51.6	57.2	51.8	16.3	4.1	1.4
	MiniCPM5-1B	84.9	77.0	81.6	74.3	43.5	3.9	1.5
	Qwen3.5-0.8B	94.3	90.2	96.8	89.1	61.7	3.0	1.2
	Qwen3.5-2B	92.0	91.6	97.4	90.4	77.6	6.2	2.1
	G9v3-3B [†]	92.2	89.7	95.8	88.2	68.3	10.2	2.3

Table 4: Main results (validated metrics, %). **Bold** = best per metric within each mode. FSP = Function Selection Performance; ACS = Argument Completeness Score; AVC = Argument Value Correctness; Task Acc = F_1 -based task accuracy; CR = strict exact-match Correct Ratio. [†]3B model (Q4_K_M), included as a reference point outside the core sub-2B study. Latency and RAM are per-query averages. 95% bootstrap confidence intervals are provided in Table 10 (Appendix).

reduction in silent parameter errors that could lead to erroneous tool calls. We note that dual-agent Task Accuracy marginally declines for Qwen3.5-2B (92.0% $\rightarrow$ 90.4%), alongside a slight regression in Function Selection Performance (96.5% $\rightarrow$ 92.0%), because routing mistakes lock in an unrecoverable incorrect tool choice; nevertheless, this trade-off remains favorable given the substantial +17.4 pp gain in fully executable, error-free calls (CR). (b) Enabling smaller models: Qwen3.5-0.8B improves from 79.8% to 89.1% Task Accuracy (+9.3 pp) while almost halving latency (5.8 s to 3.0 s; Qwen3.5-2B similarly reduces from 12.0 s to 6.2 s due to concise specialist prompts). However, latency gains are non-uniform: models that degenerate or abort early in single-agent mode (e.g., EXAONE at 0.1 s) or larger models (G9v3-3B at 10.2 s) incur higher latency under two sequential passes. (3) The 3B reference model G9v3-3B is the sole model where single-agent outperforms dual-agent across both Task Accuracy and Correct Ratio metrics (90.6% vs. 88.2% Task Accuracy; 71.7% vs. 68.3% CR), consistent with a capacity threshold hypothesis beyond which decomposition provides diminishing returns.

7.2 Per-Tool-Group Analysis

Table 5 shows per-group accuracy for the top two dual-mode models; the full breakdown across all seven models under both architectures is given in Appendix H (Tables 11 and 12), with the corresponding heatmap in Figure 4. `get_top_expenses` reaches 99.4% accuracy

Group	Qwen3.5-0.8B		Qwen3.5-2B	
	T.Acc	CR	T.Acc	CR
Time Series	81.4	54.8	74.4	60.9
Distribution	88.6	53.3	94.2	80.0
Comparison	92.7	71.4	93.8	72.9
Calc. Total	87.8	59.1	90.2	79.1
Top Expenses	92.8	62.9	99.4	94.3

Table 5: Per-group results in dual mode (validated, %). T.Acc = Task Accuracy, CR = Correct Ratio.

with Qwen3.5-2B due to its simple parameter structure, aligning with its role as one of the most common user questions (“what were my biggest purchases?”). Conversely, `plot_time_series` is the most difficult task (74.4% for Qwen3.5-2B) due to complex date specifications and relative time expressions (e.g., “past X months”) requiring temporal reasoning. G9v3-3B dual follows a similar pattern, achieving 96.1% on distribution but only 67.4% on time series.

7.3 Complexity Tier Analysis

Figure 2 presents the per-model accuracy breakdown across complexity tiers in dual mode. Task Accuracy decreases from 98.5% (L1) to 81.8% (L3) for Qwen3.5-2B, and from 95.2% to 84.2% for Qwen3.5-0.8B. G9v3-3B dual degrades similarly (96.0% L1, 81.6% L3). L3 queries involve constructs like cross-year range comparisons (“compare groceries nov 2024 to april 2025 vs nov 2025 to april 2026”), which require extracting up to 10 parameters with complex temporal constraints, similar to the kind of complicated multi year period over period analysis users commonly want.

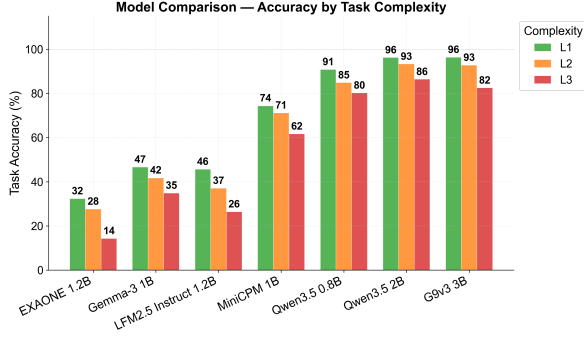


Figure 2: Task Accuracy by complexity tier (L1/L2/L3) per model in dual mode. The Qwen3.5 family degrades gracefully with increasing complexity.

7.4 Error Taxonomy Analysis

Figure 3 visualises the full error distribution across all evaluated models and architectures. The error distribution shifts markedly between configurations (RQ3).

For Qwen3.5-2B: single-agent produces 22.6% PARAM_EXTRA and 7.3% PARAM_MISSING; dual-agent reduces PARAM_EXTRA to 1.9% while PARAM_MISSING remains at 8.0%. We interpret this shift as a potentially favorable error profile change, though we note this interpretation has not been validated with formal user studies: a PARAM_EXTRA error silently restricts query scope (e.g., adding a spurious months=1 returns last month’s total instead of all-time), producing a plausible but wrong answer that users may accept without noticing the truncation (Avižienis et al., 2004; Li et al., 2023). A PARAM_MISSING error yields a visibly broader result, which we hypothesize is more likely to prompt user scrutiny. The dual-agent shift from extra to missing parameters thus represents a potentially favorable change in error profile. Qwen3.5-0.8B shows a similar pattern: single-agent produces 32.5% PARAM_MISSING / 11.8% PARAM_EXTRA; dual-agent reduces these to 12.9% / 11.1%. G9v3-3B is a notable exception: PARAM_EXTRA remains at $\approx 11\%$ in both architectures (11.1% single, 11.8% dual).

7.5 Token Efficiency and Validation

Despite using $\approx 2,700$ prompt tokens per query (vs. $\approx 5,500$ for single-agent), the dual-agent architecture achieves equivalent or superior accuracy (see Figure 7 in the Appendix). The single-agent pipeline must simultaneously attend to all five tool schemas, while the dual-agent decomposes the context, focusing each pass on a single subtask, achieving higher accuracy per token, an impor-

tant property for memory-constrained on-device deployment. The post-hoc validation layer (Section 4.3) contributes modest but consistent gains (up to +0.9 pp Task Accuracy for Qwen3.5-0.8B), primarily correcting category casing and morphological normalisation. This indicates that category name normalisation is effectively handled by lightweight fuzzy matching, whereas semantic argument extraction (temporal filters, ranges, and tool selection) remains the primary performance bottleneck.

8 Discussion

Privacy through Task Decomposition. Dual-agent decomposition improves strict correctness and, for most models, latency, making local inference more practical as a privacy guarantee. By separating routing (single digit) from extraction (focused JSON), dual-agent reduces generation complexity per call. For the Qwen3.5 family this yields large CR gains (+21.5 pp for Qwen3.5-0.8B, +17.4 pp for Qwen3.5-2B) alongside substantial latency reductions. However, this doesn’t lead to improvements across all metrics uniformly: for Qwen3.5-2B, the CR gain comes with a modest Task Accuracy and FSP regression (Section 5), as an incorrect routing decision is unrecoverable regardless of specialist quality, so decomposition trades a small amount of joint-metric accuracy for a much larger gain in fully-correct, executable calls. G9v3-3B is a starker case: single-agent outperforms dual-agent on *both* Task Accuracy and CR, consistent with a capacity threshold hypothesis beyond which decomposition becomes unnecessary or counterproductive.

Comparison with Cloud-Scale Tool Calling.

Our study is complementary to cloud-focused financial evaluations like FinMCP-Bench (Zhu et al., 2026) and FinTrace (Cao et al., 2026), which evaluate $\geq 7B$ models on multi-step trajectories. SpendSight targets single-turn, on-device, privacy-preserving tool calling where latency must be minimal and no data leaves the device. Our finding that Qwen3.5-0.8B achieves 89.1% Task Accuracy (61.7% CR) suggests that for single-turn personal finance queries, sub-2B models may approach performance levels typically associated with cloud-scale deployment, though direct comparison with $\geq 7B$ models on the same test suite remains necessary to substantiate this observation.

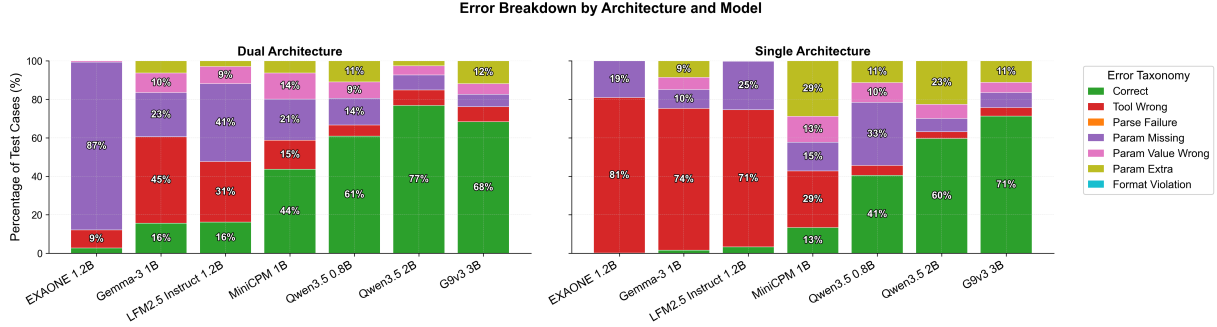


Figure 3: Error taxonomy breakdown (% of test cases) per model and architecture. Single-agent Qwen3.5-2B produces 22.6% PARAM_EXTRA errors (spurious parameters that lead to silently incorrect financial outputs). Dual-agent eliminates most of these (falling to 1.9%), consolidating residual errors into the more transparent PARAM_MISSING category.

Model Scale vs. Quantization. G9v3-3B (3B, Q4_K_M, 2.4 GB) demonstrates an alternative path: a larger model under aggressive quantization. Its 90.6% single-agent Task Accuracy matches sub-2B dual-agent accuracy at 2.3 s latency. However, dual-agent G9v3-3B incurs 10.2 s latency without error-profile improvements. These results are consistent with a capacity threshold hypothesis around 2–3B parameters beyond which single-agent capacity suffices (Kavathekar et al., 2025), though we present this as a hypothesis rather than an established finding, since it is based on a single 3B model at a different quantization scheme (Q4_K_M vs. Q8_0) and requires validation against additional 2–4B models at matched quantization before it can be generalized.

Addressing Highly Complex Queries. Even the strongest configurations achieve limited strict Correct Ratio on L3 queries: 59.1% for Qwen3.5-2B dual and 59.6% for the 3B reference G9v3-3B dual (Table 14 in Appendix H; see also Figure 2), which involve complex cross-year comparisons requiring up to 10 parameters. Fine-tuning on domain-specific synthetic data (Liu et al., 2024c,a) could improve performance on L3 queries while preserving on-device deployment.

Deployment Illustration. To illustrate end-to-end on-device execution in a realistic personal finance setting, a local deployment tracked a participant’s snack expenses over six months using dual-agent Qwen3.5 (Figure 8 in Appendix E). A period comparison query (Jul–Dec 2025 vs. Jan–Jun 2026) executed locally in 8.7 s, reflecting an 84.7% spending reduction (¥38,453 → ¥5,893) with all queries and personal ledger records remaining strictly on-device. We report this purely as a qualitative demonstration of system feasibility;

with $n = 1$, no causal claim regarding behavioural change is made.

9 Conclusion

We presented SpendSight, a study of privacy-preserving on-device personal finance Q&A tool calling with quantized local LLMs. For the Qwen3.5 family, local inference is practically viable: Qwen3.5-0.8B dual achieves 89.1% Task Accuracy (61.7% CR) at 3.0 s latency on-device. Dual-agent decomposition improves strict correctness (60.2% → 77.6% CR for Qwen3.5-2B) and shifts the error profile from silent PARAM_EXTRA toward visible PARAM_MISSING failures, while G9v3-3B reaches 90.6% Task Accuracy (71.7% CR) without decomposition. A deployment illustration demonstrates end-to-end on-device functionality in a personal expense tracking workflow without data leaving the device. We recommend Qwen3.5-0.8B dual for memory-constrained settings (≤ 2 GB) (89.1% Task Accuracy and 61.7% CR at 3.0 s), and G9v3-3B single for minimal-latency single-pass deployment (90.6% Task Accuracy, 71.7% CR at 2.3 s).

Limitations

There are several limitations to this study: (1) The architectural comparison is partially confounded: dual-agent prompts differ from single-agent not only in task decomposition but also in prompt length ($\approx 2,700$ vs. $\approx 5,500$ tokens), schema exposure, and few-shot example specificity. An equal-token-budget single-agent control and an oracle-router specialist-only ablation would isolate decomposition from prompt-engineering effects. (2) No ≥ 7 B or cloud-hosted model was evaluated on the same 115 test cases, so observations regarding parity with cloud-scale models remain indicative

rather than definitively established. (3) Test cases were authored by a single developer in English reflecting a Japan-based context, and few-shot examples exhibit substantial structural overlap with test cases (Appendix J); an independently authored held-out test set is necessary to evaluate broader out-of-distribution generalization. (4) Sub-2B models use Q8_0 quantization while G9v3-3B uses Q4_K_M, confounding the interaction between parameter scale and quantization precision, and bearing directly on the capacity-threshold hypothesis in Section 8. (5) Dual-agent is evaluated purely as an inference-time strategy on off-the-shelf GGUF weights; comparison against domain-specific SFT or RL post-training baselines (Liu et al., 2024c,a; Chen and Li, 2024) was not conducted. (6) Closed-world assumption: all queries are mapped to one of five supported tools, without evaluating out-of-domain rejection, conversational fallback, or clarification requests. (7) Evaluation measures tool selection and argument correctness against deterministic parameter schemas rather than end-to-end ledger SQL execution or UI chart rendering. (8) Ground-truth labels are deterministic and rule-derived rather than reflecting ambiguity in real user intent or inter-annotator agreement. (9) Privacy guarantees are architectural (local on-device processing) rather than cryptographic; side-channel leakage or local device compromise fall outside the threat model.

Ethics Statement

SpendSight preserves user financial privacy via on-device inference. The category taxonomy (Appendix B) and illustrative deployment (Section 8) derive from author self-tracked expense data, used with consent. Evaluation queries are synthetic; no real transaction amounts or records are in the test suite. No third-party data was used. The system is intended to complement professional financial advice. We advocate for clear user disclaimers in production.

Data and Code Availability

Code, prompts, benchmark data, and full results are publicly available at <https://github.com/SunTsunami/SpendSight-benchmark>. Released artefacts include: all 115 test cases with ground-truth labels, the complete router and specialist prompt templates, inference configuration details, raw and validated CSV result files, and the benchmark runner, plotting scripts, and the

exact model GGUFs used in this study to ensure complete reproducibility.

Acknowledgements

The authors are grateful to Sukrit Mittal for his detailed feedback on an early draft, which helped improve the clarity and rigour of this work. His suggestions on tightening the abstract, formalising the writing style, and strengthening the privacy framing helped shape the final paper.

References

- Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, Alon Benhaim, Misha Bilenko, Sebastien Bjorck, and 1 others. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.
- AI9Stars. 2026. G9v3-3B. <https://huggingface.co/ai9stars/G9v3-3B>. Accessed: 2026-08-04.
- Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. 2004. **Basic concepts and taxonomy of dependable and secure computing**. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33.
- Margaret Blake and Jo d’Ardenne. 2012. **Developing expenditure questions: Findings from rl cognitive testing**. Technical Report WP1219, Institute for Fiscal Studies.
- Yupeng Cao, Haohang Li, Weijin Liu, Wenbo Cao, Anke Xu, Lingfei Qian, Xueqing Peng, Minxue Tang, Zhiyuan Yao, Jimin Huang, K. P. Subbalakshmi, Zining Zhu, Jordan W. Suchow, and Yangyang Yu. 2026. FinTrace: Holistic trajectory-level evaluation of LLM tool calling for long-horizon financial tasks. *arXiv preprint arXiv:2604.10015*.
- Wei Chen and Zhiyuan Li. 2024. Octopus v2: On-device language model for super agent. *arXiv preprint arXiv:2404.01744*.
- Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. 2021. **FinQA: A dataset of numerical reasoning over financial data**. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3697–3711.
- B. Efron. 1979. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26.
- Gemma Team. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.
- Georgi Gerganov. 2023. [llama.cpp](https://github.com/ggerganov/llama.cpp).
- Tom Gunter, Zirui Wang, Chong Wang, Ruoming Pang, Andy Narayanan, Aonan Zhang, and Apple Intelligence Team. 2024. Apple intelligence foundation language models. *arXiv preprint arXiv:2407.21075*.
- Young-ae Hahn, Jiyou Lee, and Minhyuk Kim. 2022. **Envisioning personal finance and expense tracking for a sustainable future**. In *Proceedings of the Design Research Society 2022 Conference*.
- Annette Jäckle, Carina Cornesse, Alexander Wenz, and Mick P. Couper. 2024. **Measuring expenditure with a mobile app: Do probability-based and nonprobability panels differ?** *Journal of Survey Statistics and Methodology*, 12(5):1224–1253.
- Polaris Jhandi, Owais Kazi, Shreyas Subramanian, and Neel Sendas. 2025. Small language models for efficient agentic tool calling: Outperforming large models with targeted fine-tuning. *arXiv preprint arXiv:2512.15943*.
- Ishan Kavathekar, Raghav Donakanti, Ponnuramam Kumaraguru, and Karthik Vaidhyanathan. 2025. **Small models, big tasks: An exploratory empirical study on small language models for function calling**. In *Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering*, pages 1117–1126.
- Yaron Levi. 2026. **Personal financial information presentation and consumer spending**. *Journal of Financial and Quantitative Analysis*.
- LG AI Research. 2025. EXAONE 4.0: Unified large language models integrating non-reasoning and reasoning modes. *arXiv preprint arXiv:2507.11407*.
- Jinyang Li, Binyuan Hui, Ge Qu, Jiayi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C.C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM already serve as a database interface? A big bench for large-scale database grounded text-to-SQLs. In *Advances in Neural Information Processing Systems*, volume 36, pages 42330–42357.
- Yuanchun Li, Hao Wen, Weijun Wang, Weijun Wang, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, Rui Kong, Yile Wang, Hanfei Geng, Jian Luan, Xuefeng Jin, Zilong Ye, Guanqing Xiong, Fan Zhang, Xiang Li, and 5 others. 2024. Personal LLM agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459*.
- Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, Jun Wang, and Weinan Zhang. 2024. Hammer: Robust function-calling for on-device language models via function masking. *arXiv preprint arXiv:2410.04587*.
- Liquid AI. 2026. **Introducing LFM2.5: The next generation of on-device AI**. Liquid AI Blog.
- Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, and 8 others. 2024a. ToolACE: Winning the points of LLM function calling. *arXiv preprint arXiv:2409.00920*.
- Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, Liangzhen Lai, and Vikas Chandra. 2024b. **MobileLLM: Optimizing sub-billion parameter language models for on-device use cases**. In *Proceedings of the 41st International Conference on Machine Learning*, pages 32431–32454.

- Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Tan, Joey Tianyi Zhou, Philip S. Yu, and Steven C. H. Hoi. 2024c. APIGen: Automated pipeline for generating verifiable and diverse function-calling datasets. *Advances in Neural Information Processing Systems*, 37.
- Helen Nissenbaum. 2004. Privacy as contextual integrity. *Washington Law Review*, 79(1):119–158.
- OpenBMB. 2026. [MiniCPM5-1B: A SOTA 1B on-device LLM](#).
- Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. [The Berkeley function calling leaderboard \(BFCL\): From tool use to agentic evaluation of large language models](#). In *Proceedings of the 42nd International Conference on Machine Learning*, pages 48371–48392.
- Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. [Gorilla: Large language model connected with massive APIs](#). *arXiv preprint arXiv:2305.15334*.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. [ToolLLM: Facilitating large language models to master 16000+ real-world APIs](#). In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR 2024)*.
- Qwen Team. 2026. [Qwen3.5: Towards native multi-modal agents](#). Qwen Blog.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36.
- Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. HuggingGPT: Solving AI tasks with chatgpt and its friends in Hugging Face. In *Advances in Neural Information Processing Systems*, volume 36.
- Yewei Song, Xunzhu Tang, Cedric Lothritz, Saad Ezzini, Jacques Klein, Tegawendé F. Bissyandé, Andrey Boytsov, Ulrick Ble, and Anne Goujon. 2025. [CallNavi: A challenge and empirical study on LLM function calling and routing](#). In *Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering*.
- Robin Staab, Mark Vero, Mislav Balunovic, and Martin Vechev. 2024. Beyond memorization: Violating privacy via inference with large language models. In *Proceedings of the 12th International Conference on Learning Representations*.
- Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.
- Jiajun Xu, Zhiyuan Li, Wei Chen, Qun Wang, Xin Gao, Qi Cai, and Ziyuan Ling. 2024. On-device language models: A comprehensive review. *arXiv preprint arXiv:2409.00088*.
- Hongyang Yang, Xiao-Yang Liu, and Christina Dan Wang. 2023. [FinGPT: Open-source financial large language models](#). In *International Symposium on Large Language Models for Financial Services (FinLLM 2023) at IJCAI 2023*.
- Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (LLM) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, 4(2):100211.
- Jie Zhu, Yimin Tian, Boyang Li, Kehao Wu, Zhongzhi Liang, Junhui Li, Xianyin Zhang, Lifan Guo, Feng Chen, Yong Liu, and Chi Zhang. 2026. [FinMCP-Bench: Benchmarking LLM agents for real-world financial tool use under the model context protocol](#). *arXiv preprint arXiv:2603.24943*.

A Complexity Scoring Features

Feature	Range	Description
n_p	0+	Number of expected parameters
d_{date}	0–4	Date complexity: none / year / month / range / specific day
d_{cat}	0–2	Category distance: exact / alias / semantic
d_{rel}	0–1	Relative time expression (“past X months”)
d_{multi}	0–2	Compound parameter groups
d_{abbr}	0–1	Abbreviated year formats (“’24”)
d_{hol}	0–2	Holiday/calendar knowledge required

Table 6: Seven features of the complexity scoring function (Eq. 1).

B Known Category Vocabulary

The post-hoc parameter validation layer (Section 4.3) and the LLM prompts rely on a known category vocabulary defined dynamically or in static schema files. Table 7 lists the 11 major categories defined in the application alongside the mapped subcategories/aliases from the system configuration (`categories.json`) that are passed in the `{metadata}` block.

Major Category	Mapped Minor Categories / Subcategories
Housing and Utilities	housing, internet bill, electricity bill, gas bill, water & sewage bill, phone bill
Food	grocery, snacks, cafe, café, coffee, bento, beverage, combini meal, dining, eating with friend
Transportation	commute, ride share, tokyo metro, Tokyo Metro, flight tickets, cable car, bus, shinkansen, car rental, taxi
Accommodation	stay
Fitness	supplements, shoes, sports event, sports watch, sports clothing, sports rental, gym, sports equipment, basketball game, football game, futsal game
Souvenirs/Gifts/Treats	souvenirs, treat, gift, souvenirs/gifts/treats
Household and Clothing	clothing, household
Entertainment	entertainment, nomikai, activities, Activities, arcades & karaoke, Arcades & Karaoke, events & venues, Events & Venues
Miscellaneous	medicines, personal care, misc, help, charity, donation, entrance fees, park entrance fees, healthcare
Education	tuition, books, exam fees, notebooks, textbook
Electronics and Furniture	furniture, electronics

Table 7: Known category vocabulary from `categories.json`.

C Sample Test Cases

ID	Query	Tool	Expected Params	Tier
TS01	Can ya show spending on food for past 6 months?	<code>plot_time_series</code>	<code>category=Food, months=6</code>	L2
DI06	show breakdown of expenses for june 21 2024	<code>plot_distribution</code>	<code>year=2024, month=6, day=21</code>	L2
CP24	compare groceries nov 2024 to apr 2025 vs nov 2025 to apr 2026	<code>plot_comparison_bars</code>	<code>category=grocery, y1=2024, sm1=11, ey1=2025, em1=4, y2=2025, sm2=11, ey2=2026, em2=4</code>	L3
CT07	sum total spent on metro for july 2024?	<code>calculate_total</code>	<code>category=tokyo metro, year=2024, month=7</code>	L2
TP06	top 5 expenses past month? disregard rent	<code>get_top_expenses</code>	<code>n=5, months=1, ignore_rent=true</code>	L2

Table 8: Sample test cases. Note informal language, abbreviations, and semantic mappings (“metro”→“tokyo metro”, “groceries”→“grocery”).

D Qualitative Error Examples

Table 9 illustrates representative failures for the dominant error types in Qwen3.5-2B (dual mode).

Error Type	Query	Expected	Model Output
PARAM_MISSING	“Show spending trend for the last 3 months”	{months: 3}	{ } (months omitted in the relative-time case)
PARAM_VALUE_WRONG	“compare groceries nov 2024 to apr 2025 vs nov 2025 to apr 2026”	{em1: 4, em2: 4}	{em1: 11, em2: 11} (confuses start month for end month in cross-year range)
PARAM_EXTRA	“How much did I spend on food?”	{category: Food}	{category: Food, months: 1} (spurious time filter restricts scope to last month, silently returning an incorrect total)

Table 9: Representative failure cases for Qwen3.5-2B (dual mode). `PARAM_EXTRA` is the most dangerous failure mode in financial contexts: the spurious `months=1` filter silently restricts “total food spending” to “last month’s food spending”, returning a plausible but incorrect figure that a user may act on without realising the scope was changed.

E Supplementary Figures

Figure 4 presents the model $\times$ category Task Accuracy heatmap. Figure 5 shows the per-category footprint across all evaluated models. Figure 6 illustrates the metrics improvement provided by the post-hoc validation layer. Figure 7 shows the token efficiency trade-off between architectures. Figure 8 illustrates the period comparison interface from the on-device deployment illustration (Section 8).

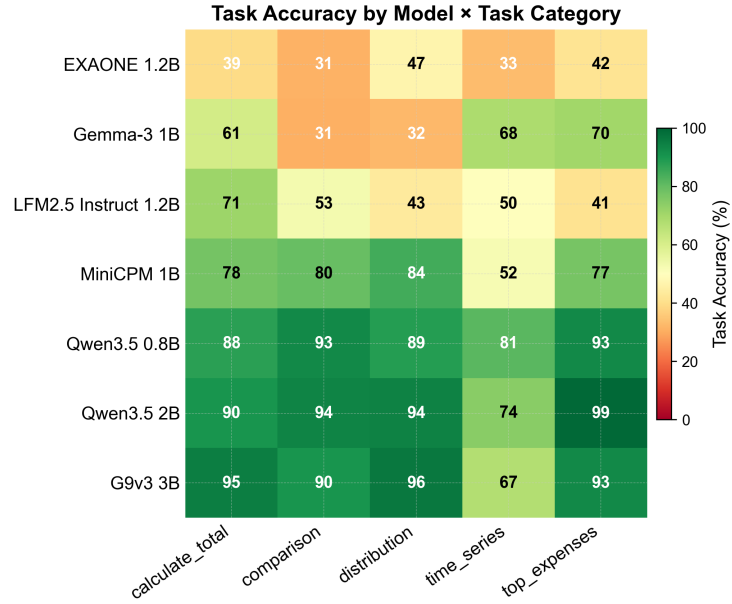


Figure 4: Task Accuracy (%) by model $\times$ tool category in dual mode. Qwen3.5 models achieve $>74\%$ across all categories, while weaker models show marked gaps in comparison and time series tasks.

Per-Model Task Footprint: Architecture Comparison

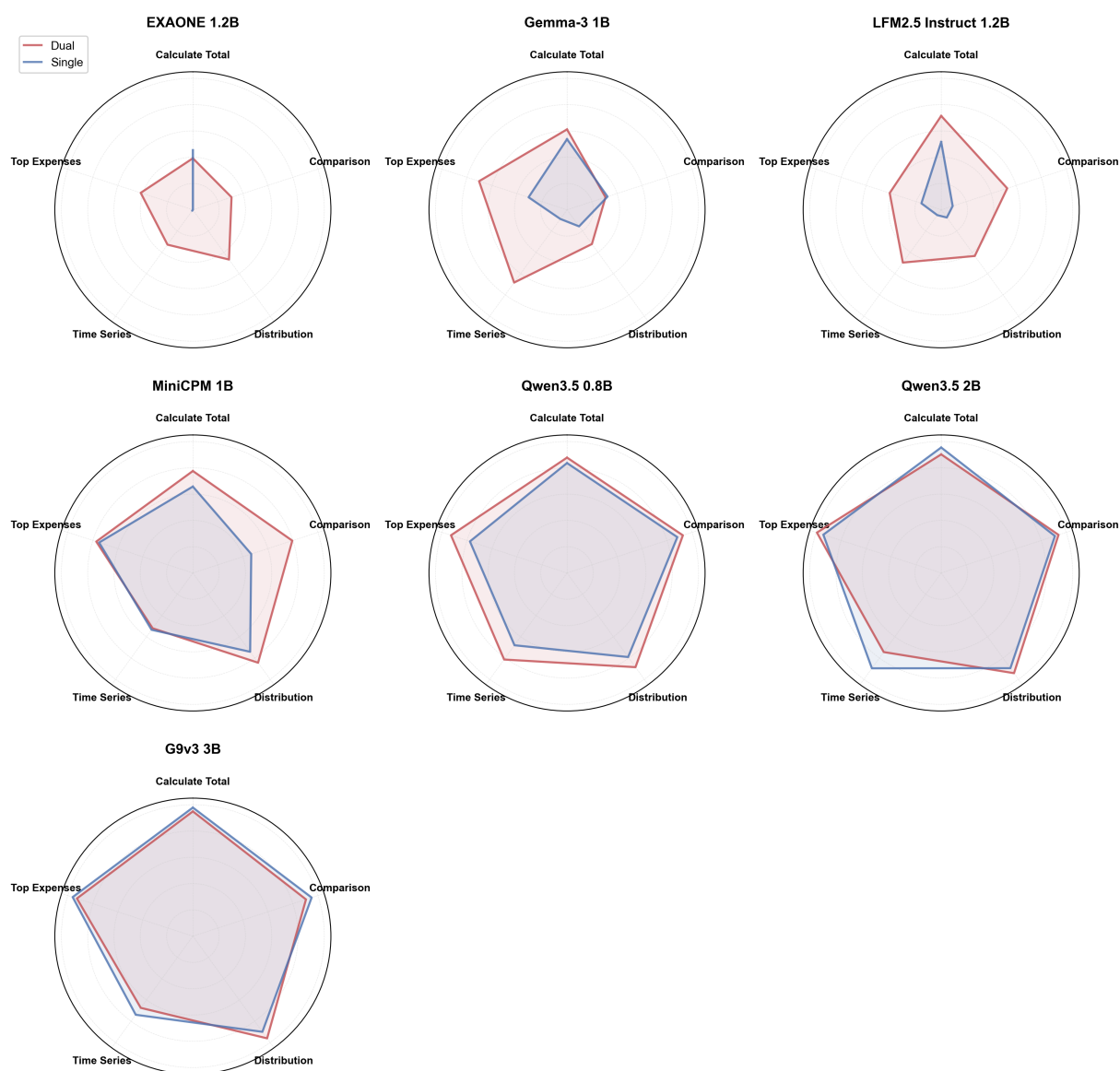


Figure 5: Per-model task footprint across all five task categories (Single vs. Dual mode). The dual-agent architecture provides near-universal improvement across models and tasks.

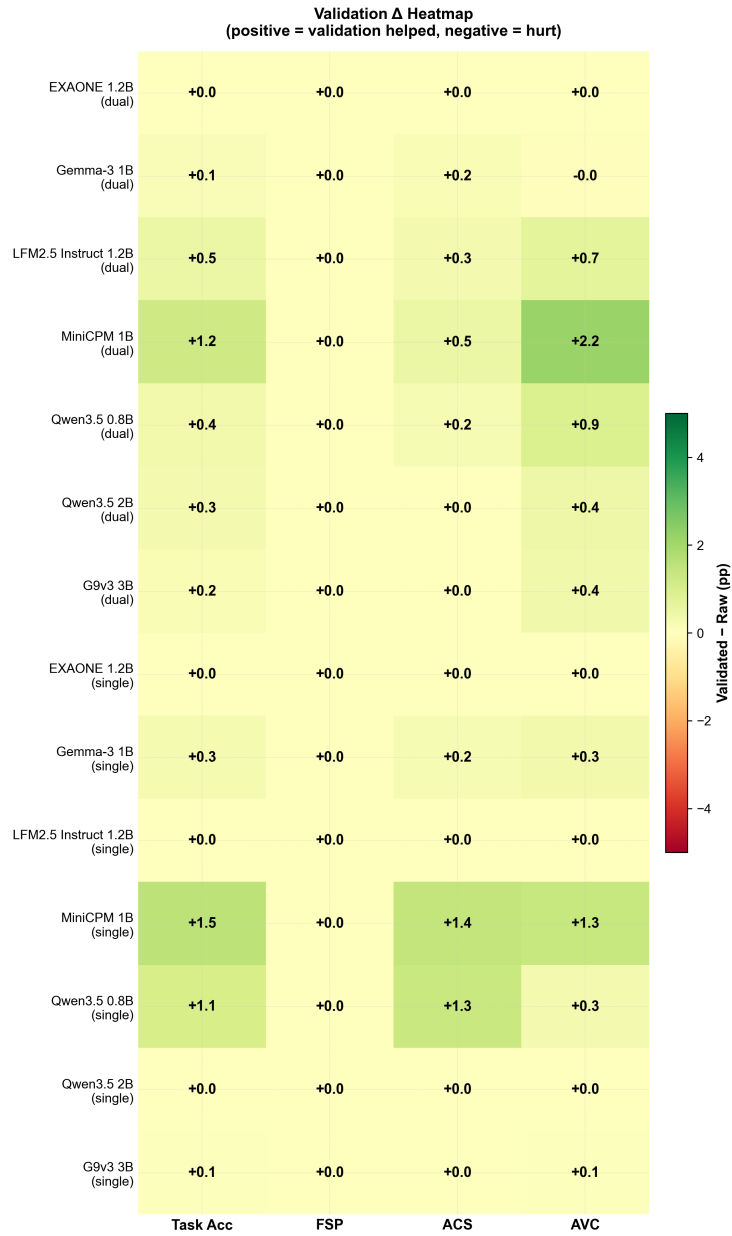


Figure 6: Validation Δ Heatmap, showing absolute percentage point improvement across metrics after applying post-hoc fuzzy matching.

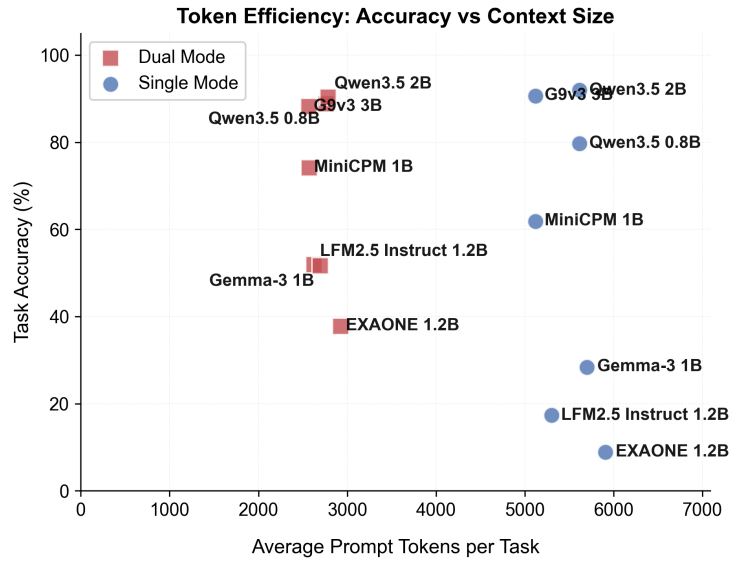


Figure 7: Token efficiency: Task Accuracy vs. average prompt tokens per query. Dual-agent configurations (red squares) achieve higher accuracy with fewer total tokens than their single-agent counterparts (blue circles), demonstrating superior context utilisation.

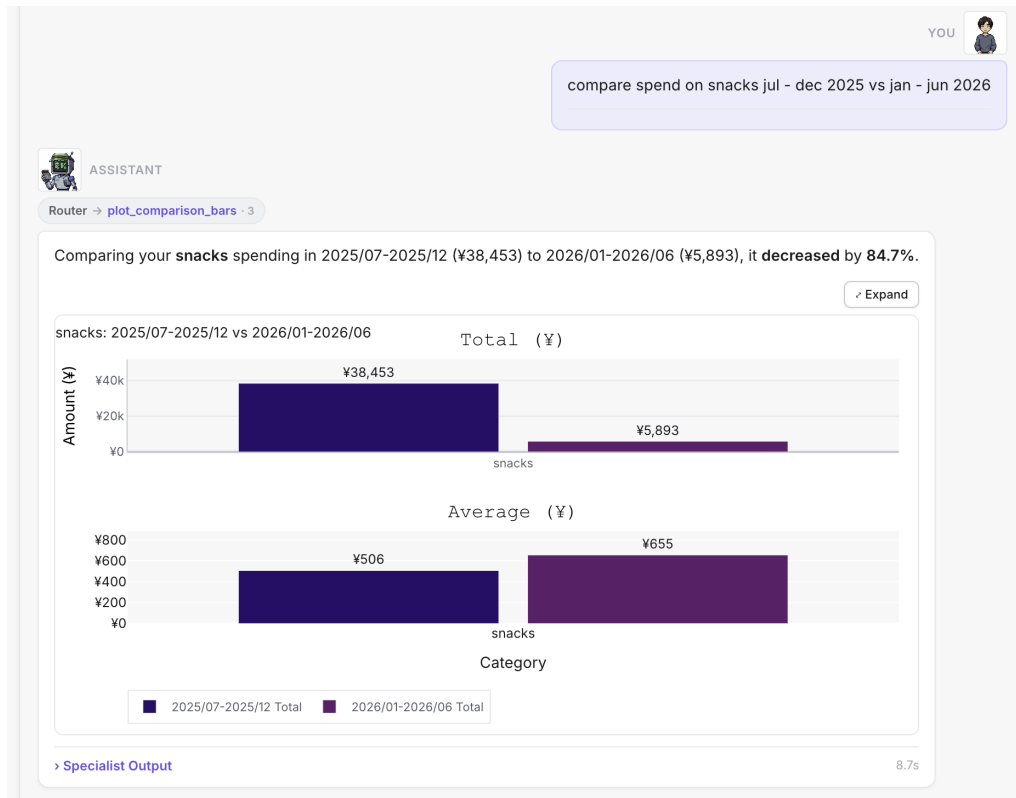


Figure 8: Illustrative deployment: period comparison view (Jul–Dec 2025 vs. Jan–Jun 2026) generated locally by Qwen3.5-2B dual in 8.7 s, showing an 84.7% reduction in snack spending (¥38,453 → ¥5,893). Presented as a qualitative demonstration of end-to-end functionality (Section 8); no causal behavioral claim is made ($n = 1$).

F Prompt Templates

The on-device prompts utilised for routing and parameter extraction tasks are detailed below.

F.1 Router Prompt

The router receives a focused intent-classification prompt:

You are an expert intent classifier for an expense analysis assistant.
Your job is to determine which ONE tool is best suited to answer the user's question.

Available Tools

1. `plot_time_series` (ID: 1)
 - Use when user asks about: trends, spending over time, "past X months", "since 2023", date ranges.
 - Keywords: trend, over time, plot, show spending, changed, since.
 - NOTE: if user says "trend" or "over time" WITH "exclude rent", still use this tool.
2. `plot_distribution` (ID: 2)
 - Use when user asks for: breakdown, distribution, proportions, pie chart, "how is X split", "what does X look like".
 - Keywords: breakdown, distribution, distro, pie chart, split, look like.
 - NOTE: if user says "breakdown" or "split" WITH "exclude rent", still use this tool.
3. `plot_comparison_bars` (ID: 3)
 - Use ONLY when comparing TWO different time periods (e.g. "Dec 2024 vs Dec 2025" or ranges like "Jan-Jun 2024 vs Jan-Jun 2025").
 - REQUIRES two distinct periods. If the user only mentions one month or one year, DO NOT use this tool.
 - Keywords: compare, contrast, vs, versus, difference between.
4. `calculate_total` (ID: 4)
 - Use when user asks for: simple totals, sums, specific amounts "how much did I spend".
 - Keywords: how much, total, sum, cost.
5. `get_top_expenses` (ID: 5)
 - Use when user asks for: biggest/largest expenses, top X items, most expensive.
 - Keywords: biggest, largest, top, most expensive, highest.
 - NOTE: if user says "top X" WITH "exclude rent", still use this tool.

Output Format

Output ONLY the tool number (1-5). Do not output anything else, NO MARKDOWN, NO BACKTICKS, NO EXPLANATION, NO CODE!

WRONG outputs (NEVER do this):

```
[X] I think the best tool is 4
[X] `4`
[X] Based on the query, I would use 1 because...
[X] ```4```
[X] The answer is: 2
```

CORRECT outputs:

```
[v] 4
[v] 1
```

Examples

User: "How much did I spend on food?"

Output: 4

User: "Total cost of internet for the last 3 months"

Output: 4

User: "How much did I spend in the last 6 months without rent?"

Output: 4

User: "Show me my spending trend for groceries"

Output: 1

User: "How has my rent changed over the last 6 months?"

Output: 1

User: "Show spending trend for past 4 months exclude rent tho"

Output: 1

User: "Compare food spending in 2024 and 2025"

Output: 3

User: "Difference in spending between Dec 2023 and Dec 2024"

Output: 3

User: "Compare overall spend in '24 vs '25 (exclude rent tho)"

Output: 3

User: "Compare dining between Jan-Jun 2024 and Jan-Jun 2025"
Output: 3

User: "Contrast spending from Nov 2024 to Feb 2025 vs Nov 2025 to Feb 2026"
Output: 3

User: "What were my biggest expenses?"
Output: 5

User: "Show me my top 5 largest purchases this year"
Output: 5

User: "Top 10 items without rent"
Output: 5

User: "Pie chart of transportation"
Output: 2

User: "How is my spending split in 2023?"
Output: 2

User: "Show me my spending breakdown excluding rent for the past 6 months"
Output: 2

User: "what does my electronics spending look like for jan 2025?"
Output: 2

User: "share distro of household expenses"
Output: 2

F.2 Specialist Base Template

Each specialist prompt instantiates a shared template defining output structure and relative date handling. The `{metadata}` placeholder is populated dynamically with the categories present in the user-uploaded dataset:

You are an automated API parameter extractor. Your ONLY purpose is to extract parameters from the user's request and output a SINGLE JSON object containing those parameters.

```
## CRITICAL RULES (READ CAREFULLY)
1. DO NOT write any Python code or function calls. DO NOT import anything.
2. DO NOT wrap the output in markdown. NO backticks (`).
3. Output EXACTLY and ONLY a JSON object representing the parameters.
   E.g. {"category": "Food", "months": 6}
4. Use EXACT category names from the metadata below. If not an exact match,
   map it to the closest valid category (e.g. 'groceries' -> 'grocery',
   'dining out' -> 'dining').

## Date Rules
- 'months=N': RELATIVE duration (e.g. "past month" -> months=1, "last 3
  months" -> months=3, "past year" -> months=12).
- Fractional years: convert to months. "past 0.5 years" -> months=6, "past
  2.5 years" -> months=30.
- Abbreviated years: '24 = 2024, 24 = 2024, ALWAYS output the full 4-digit
  year.
- 'year=YYYY': specific full calendar year.
- 'year=YYYY, month=M': specific calendar month (e.g. "Jan 2024").
- 'year=YYYY, month=M, day=D': specific calendar date.
- 'start_year=YYYY, end_year=YYYY': specific year range.
- 'start_year=YYYY, start_month=M, end_year=YYYY, end_month=M': specific
  month-to-month range.
- For comparing two ranges (e.g., comparing Jan-Jun 2024 vs Jan-Jun 2025),
  use comparison range parameters: 'y1=2024, sm1=1, em1=6, y2=2025, sm2=1,
  em2=6'.
- For comparing cross-year ranges (e.g., Nov 2024-Jan 2025 vs Nov 2025-Jan
  2026), use 'ey1' and 'ey2': 'y1=2024, sm1=11, ey1=2025, em1=1, y2=2025,
  sm2=11, ey2=2026, em2=1'.
- CRITICAL: For relative queries like "past month", "last month", "last 6
  months", ALWAYS use 'months=N'. Do NOT use 'month=M'.
- Pick ONLY ONE time filter. Do not mix 'months' with 'year'.

## Available Parameters
{parameters}

## Examples
{examples}

## Context
```
{metadata}
```
Currency: JPY
Today: {current_date}
```

FINAL REMINDER: Output ONLY the JSON object. NO MARKDOWN, NO BACKTICKS, NO EXPLANATION, NO CODE!

F.3 Specialist Tool Prompts

Each tool-specific specialist appends its unique parameter definitions and few-shot examples to the base template. The precise specifications for each tool are detailed below.

F.3.1 `plot_time_series`

```
Parameters:
  category (str, optional), year (int, optional), month (int, optional),
  start_year (int, optional), start_month (int, optional),
  end_year (int, optional), end_month (int, optional),
  months (int, optional), ignore_rent (bool, optional, defaults to false)

## Examples
Q: "How much did I spend on futsal for the past 6 months?"
{"category": "futsal game", "months": 6}

Q: "Show me food spending from 2023 to 2025"
{"category": "Food", "start_year": 2023, "end_year": 2025}

Q: "Plot spending on snacks from oct 2024 to dec 2025"
{"category": "snacks", "start_year": 2024, "start_month": 10, "end_year": 2025, "end_month": 12}

Q: "Can ya plot spending on futsal on dec 2024?"
{"category": "futsal game", "year": 2024, "month": 12}

Q: "Gym expenses in 2024?"
{"category": "gym", "year": 2024}

Q: "Show me my spending trend for the last 3 months"
{"months": 3}

Q: "plot spending from june 2023 to dec 2025"
{"start_year": 2023, "start_month": 6, "end_year": 2025, "end_month": 12}

Q: "Trend of electricity bills since 2022"
{"category": "electricity bill", "start_year": 2022}

Q: "How has my transportation spending changed over time?"
{"category": "Transportation"}

Q: "Show spending on combinis for past year"
{"category": "combinini meal", "months": 12}

Q: "Show spending trend for past 4 months exclude rent tho"
{"months": 4, "ignore_rent": true}

Q: "plot spend on cafes for past 2.5 yrs"
{"category": "cafe", "months": 30}

Q: "show my ride share costs from '24 to 26"
{"category": "ride share", "start_year": 2024, "end_year": 2026}

Q: "show electronics spending since 2023"
{"category": "electronics", "start_year": 2023}
```

F.3.2 `plot_distribution`

```
Parameters:
  category (str, optional), year (int, optional), month (int, optional),
  day (int, optional), start_year (int, optional), start_month (int, optional),
  end_year (int, optional), end_month (int, optional), months (int, optional),
  ignore_rent (bool, optional, defaults to false)

## Examples
Q: "Show me a breakdown of my food expenses in 2024"
{"category": "Food", "year": 2024}

Q: "Pie chart of all expenses from Nov 2024 to Feb 2025"
{"start_year": 2024, "start_month": 11, "end_year": 2025, "end_month": 2}

Q: "Show me my spending breakdown for last month"
{"months": 1}

Q: "Show me my spending breakdown for groceries for past 6 months"
{"category": "Grocery", "months": 6}

Q: "show spend distribution for 2026 feb"
{"year": 2026, "month": 2}

Q: "show breakdown of all expenses from 2024 to 2025"
{"start_year": 2024, "end_year": 2025}
```

```

Q: "Show me a breakdown of expenses for Dec 2024 (exclude rent tho)"
{"year": 2024, "month": 12, "ignore_rent": true}

Q: "Show spending breakdown for 2023/06/22"
{"year": 2023, "month": 6, "day": 22}

Q: "Spending distribution for the last 3 months"
{"months": 3}

Q: "Breakdown of transportation spending from 2022 to 2024"
{"category": "Transportation", "start_year": 2022, "end_year": 2024}

Q: "How did I split my money between categories in 2023?"
{"year": 2023}

Q: "Show me my spending breakdown excluding rent for the past 6 months"
{"months": 6, "ignore_rent": true}

Q: "Breakdown of fitness spending for 2025"
{"category": "Fitness", "year": 2025}

Q: "Breakdown of my spending since 2024"
{"start_year": 2024}

Q: "show breakdown of all expenses for mar 2026 (exclude rent tho)"
{"year": 2026, "month": 3, "ignore_rent": true}

Q: "what does my electronics and furniture spending look like for jan 2025?"
{"category": "Electronics and Furniture", "year": 2025, "month": 1}

```

F3.3 plot_comparison_bars

```

Parameters:
    category (str, optional), y1 (int, optional), m1 (int, optional),
    d1 (int, optional), y2 (int, optional), m2 (int, optional),
    d2 (int, optional), sm1 (int, optional), em1 (int, optional),
    sm2 (int, optional), em2 (int, optional), ey1 (int, optional),
    ey2 (int, optional), ignore_rent (bool, optional, defaults to false)

## Examples
Q: "Compare food spending in 2024 vs 2025"
{"category": "Food", "y1": 2024, "y2": 2025}

Q: "Compare dining Jan 2024 vs Jan 2025"
{"category": "dining", "y1": 2024, "m1": 1, "y2": 2025, "m2": 1}

Q: "How does my spend on snacks compare between jan 2024 and jan 2026?"
{"category": "snacks", "y1": 2024, "m1": 1, "y2": 2026, "m2": 1}

Q: "Compare total spending 2022 vs 2023"
{"y1": 2022, "y2": 2023}

Q: "Compare overall expenses from jan 2025 vs jan 2026"
{"y1": 2025, "m1": 1, "y2": 2026, "m2": 1}

Q: "Compare transportation on 21 July 2024 vs 21 July 2025"
{"category": "Transportation", "y1": 2024, "m1": 7, "d1": 21, "y2": 2025, "m2": 7, "d2": 21}

Q: "Compare spend on electricity 2024 vs 2025"
{"category": "electricity bill", "y1": 2024, "y2": 2025}

Q: "Compare gym spending Nov 2024 vs Nov 2025"
{"category": "gym", "y1": 2024, "m1": 11, "y2": 2025, "m2": 11}

Q: "Compare my spending in 2024 vs 2025 excluding rent"
{"y1": 2024, "y2": 2025, "ignore_rent": true}

Q: "compare education expenses '24 vs '25"
{"category": "Education", "y1": 2024, "y2": 2025}

Q: "compare household expenses for april '24 vs apr 2025"
{"category": "household", "y1": 2024, "m1": 4, "y2": 2025, "m2": 4}

Q: "Compare dining between Jan-Jun 2024 and Jan-Jun 2025"
{"category": "dining", "y1": 2024, "sm1": 1, "em1": 6, "y2": 2025, "sm2": 1, "em2": 6}

Q: "compare electricity from jan-mar '24 vs jan-mar '25"
{"category": "electricity bill", "y1": 2024, "sm1": 1, "em1": 3, "y2": 2025, "sm2": 1, "em2": 3}

Q: "Contrast overall spend from Nov 2024 to Feb 2025 vs Nov 2025 to Feb 2026"
{"y1": 2024, "sm1": 11, "ey1": 2025, "em1": 2, "y2": 2025, "sm2": 11, "ey2": 2026, "em2": 2}

Q: "compare total spend on new years eve 2024 vs for same day on 2025"
{"y1": 2024, "m1": 12, "d1": 31, "y2": 2025, "m2": 12, "d2": 31}

```

F3.4 calculate_total

Parameters:

```

    category (str, optional), year (int, optional), month (int, optional),
    day (int, optional), start_year (int, optional), start_month (int, optional),
    end_year (int, optional), end_month (int, optional), months (int, optional),
    ignore_rent (bool, optional, defaults to false)

## Examples
Q: "How much did I spend on groceries in Dec 2024?"
{"category": "grocery", "year": 2024, "month": 12}

Q: "Total spending from Oct 2024 to March 2025"
{"start_year": 2024, "start_month": 10, "end_year": 2025, "end_month": 3}

Q: "What is my total spending in 2025?"
{"year": 2025}

Q: "How much did I spend on food in past month?"
{"category": "Food", "months": 1}

Q: "Total cost of electricity in 2023"
{"category": "electricity bill", "year": 2023}

Q: "Total spent on 15 Jan 2024?"
{"year": 2024, "month": 1, "day": 15}

Q: "How much spent on rent in the last 6 months?"
{"category": "rent", "months": 6}

Q: "Sum of all transportation expenses in 2025"
{"category": "Transportation", "year": 2025}

Q: "Total spent on combini food for 2025?"
{"category": "combin meal", "year": 2025}

Q: "How much did I spend in the last 6 months without rent?"
{"months": 6, "ignore_rent": true}

Q: "Total spent on dining since 2024"
{"category": "dining", "start_year": 2024}

Q: "whats the total spend on donations from '23 to 2026?"
{"category": "donation", "start_year": 2023, "end_year": 2026}

Q: "Excluding rent, what was my total spend in 2025?"
{"year": 2025, "ignore_rent": true}

```

F.3.5 get_top_expenses

```

Parameters:
    n (int, optional, defaults to 10), category (str, optional),
    year (int, optional), month (int, optional), day (int, optional),
    start_year (int, optional), start_month (int, optional),
    end_year (int, optional), end_month (int, optional),
    months (int, optional), min_amount (int, optional),
    ignore_rent (bool, optional, defaults to false)

## Examples
Q: "What were my biggest expenses in Dec 2024?"
{"n": 10, "year": 2024, "month": 12}

Q: "Top 5 food expenses in 2024?"
{"n": 5, "category": "Food", "year": 2024}

Q: "Top 10 food expenses in june 2025?"
{"n": 10, "category": "Food", "year": 2025, "month": 6}

Q: "Top 5 expenses for 26 june 2024?"
{"n": 5, "year": 2024, "month": 6, "day": 26}

Q: "Top expenses from July 2023 to Dec 2024"
{"start_year": 2023, "start_month": 7, "end_year": 2024, "end_month": 12}

Q: "Show my top 10 expenses from 2023 to 2025"
{"n": 10, "start_year": 2023, "end_year": 2025}

Q: "Top 10 expenses of past month?"
{"n": 10, "months": 1}

Q: "What are my top expenses excluding rent?"
{"n": 10, "ignore_rent": true}

Q: "Show my top 3 largest transactions in 2023"
{"n": 3, "year": 2023}

Q: "What were the biggest expenses over the last 3 months, without rent?"
{"n": 10, "months": 3, "ignore_rent": true}

Q: "Top 7 futsal expenses for 2025?"

```

```
{"n": 7, "category": "futsal game", "year": 2025}
```

Q: "Top 5 expenses last month, exclude rent"

```
{"n": 5, "months": 1, "ignore_rent": true}
```

Q: "What were my biggest expenses since 2023?"

```
{"n": 10, "start_year": 2023}
```

Q: "show me top 10 expenses from jan 2025 to june '25"

```
{"n": 10, "start_year": 2025, "start_month": 1, "end_year": 2025, "end_month": 6}
```

Q: "tell me the top 15 expenses for 2025, ignore rent plz"

```
{"n": 15, "year": 2025, "ignore_rent": true}
```

G Bootstrap Confidence Intervals

Table 10 provides 95% bootstrap confidence intervals for the primary metrics reported in Table 4. Resampling is performed over test cases ($n = 115$), with each test case’s metrics averaged across 5 repetitions, using $n = 2000$ resamples (Efron, 1979).

Table 10: 95% bootstrap confidence intervals ($n = 2000$ resamples over 115 test cases, each averaged across 5 repetitions). Tabulated values report bootstrap sample means and 2.5th–97.5th percentiles, consistent with original sample statistics in Table 4.

Mode	Model	Task Acc	95% CI	CR	95% CI	FSP	95% CI
Single	EXAONE-4.0-1.2B	8.9	[5.6, 12.1]	0.2	[0.0, 0.5]	19.3	[12.2, 26.3]
Single	Gemma-3-1B-it	28.4	[22.8, 34.4]	1.6	[0.0, 4.0]	26.3	[19.0, 34.8]
Single	LFM2.5-1.2B	17.3	[12.6, 22.4]	3.3	[0.7, 6.8]	28.7	[20.7, 36.9]
Single	MiniCPM5-1B	61.9	[56.8, 67.2]	13.4	[8.0, 19.5]	70.6	[62.4, 78.8]
Single	Qwen3.5-0.8B	79.7	[75.7, 83.7]	40.5	[32.0, 49.0]	95.0	[90.8, 98.4]
Single	Qwen3.5-2B	91.9	[89.5, 94.1]	59.7	[51.1, 68.2]	96.5	[93.2, 99.1]
Single	G9v3-3B [†]	90.6	[86.6, 94.1]	71.7	[63.0, 79.7]	95.7	[91.3, 99.1]
Dual	EXAONE-4.0-1.2B	37.8	[34.8, 40.7]	2.8	[0.7, 5.2]	90.6	[85.0, 95.7]
Dual	Gemma-3-1B-it	51.6	[45.4, 57.7]	16.5	[10.4, 23.5]	55.1	[46.3, 64.2]
Dual	LFM2.5-1.2B	51.9	[46.7, 57.2]	16.2	[10.3, 22.6]	68.5	[60.9, 76.2]
Dual	MiniCPM5-1B	74.1	[69.2, 79.2]	43.8	[35.3, 52.7]	84.9	[78.6, 90.3]
Dual	Qwen3.5-0.8B	88.8	[85.4, 92.1]	60.9	[52.2, 69.4]	94.3	[89.7, 97.9]
Dual	Qwen3.5-2B	90.3	[86.0, 94.3]	76.9	[69.7, 84.0]	92.0	[87.0, 96.5]
Dual	G9v3-3B [†]	88.2	[83.8, 92.3]	68.3	[59.8, 76.5]	92.2	[87.0, 96.5]

H Per-Group and Per-Tier Breakdowns

Tables 11 and 12 provide per-tool-group Task Accuracy and Correct Ratio for all seven evaluated models under both architectures, extending the two-model preview in Table 5 to the full model set. Tables 13 and 14 provide the corresponding per-complexity-tier breakdowns, giving exact numerical backing for Figure 2.

Table 11: Single-agent Task Accuracy / Correct Ratio (%) by tool group, all seven models (validated). Extends the two-model preview in Table 5.

Model	Time Series		Distribution		Comparison		Calc. Total		Top Expenses	
	T.A	CR	T.A	CR	T.A	CR	T.A	CR	T.A	CR
EXAONE-4.0-1.2B	0.9	0.9	0.0	0.0	0.0	0.0	45.4	0.0	0.0	0.0
Gemma-3-1B-it	8.6	0.0	15.6	0.0	32.4	0.0	53.7	8.2	30.8	0.0
LFM2.5-1.2B	5.0	4.3	7.3	0.0	9.2	0.0	51.6	12.7	15.8	0.0
MiniCPM5-1B	53.5	5.2	74.1	14.3	46.7	23.6	65.8	1.8	75.0	20.0
Qwen3.5-0.8B	68.0	35.7	79.1	39.0	88.2	55.7	83.6	53.6	77.7	13.3
Qwen3.5-2B	89.7	66.1	89.6	47.6	91.0	48.6	95.5	74.5	94.3	63.8
G9v3-3B [†]	73.9	56.5	90.0	54.3	95.0	74.3	97.8	87.3	96.1	85.7

Table 12: Dual-agent Task Accuracy / Correct Ratio (%) by tool group, all seven models (validated). Extends the two-model preview in Table 5; recomputed directly from the released per-observation run data, and used to correct that preview table for consistency with Table 4 and Table 10.

Model	Time Series		Distribution		Comparison		Calc. Total		Top Expenses	
	T.A	CR	T.A	CR	T.A	CR	T.A	CR	T.A	CR
EXAONE-4.0-1.2B	32.8	2.6	46.8	1.0	30.9	0.0	39.2	2.7	41.7	8.6
Gemma-3-1B-it	68.5	39.1	32.2	4.8	30.8	3.6	61.2	18.2	70.4	19.0
LFM2.5-1.2B	49.7	6.1	43.5	17.1	52.8	19.3	71.4	37.3	41.2	0.0
MiniCPM5-1B	52.0	19.1	84.4	63.8	79.5	45.7	77.6	44.5	77.3	47.6
Qwen3.5-0.8B	81.4	54.8	88.6	53.3	92.7	71.4	87.8	59.1	92.8	62.9
Qwen3.5-2B	74.4	60.9	94.2	80.0	93.8	72.9	90.2	79.1	99.4	94.3
G9v3-3B [†]	67.4	42.6	96.1	77.1	90.5	64.3	94.9	75.5	92.8	85.7

Table 13: Single-agent Task Accuracy and CR by complexity tier (validated, %). Companion to Table 14 and Figure 2.

Model	L1		L2		L3	
	T.A	CR	T.A	CR	T.A	CR
EXAONE-4.0-1.2B	16.7	0.0	12.2	0.4	1.5	0.0
Gemma-3-1B-it	38.4	0.0	27.9	4.0	23.6	0.0
LFM2.5-1.2B	29.1	3.3	20.1	6.7	8.5	0.0
MiniCPM5-1B	69.7	12.5	66.2	10.2	53.6	17.0
Qwen3.5-0.8B	86.5	45.8	79.7	39.1	76.2	39.1
Qwen3.5-2B	93.9	73.3	91.8	56.0	91.0	56.1
G9v3-3B [†]	96.7	80.0	94.8	74.2	83.4	64.8

Table 14: Dual-agent Task Accuracy and CR by complexity tier (validated, %). Companion to Table 13 and Figure 2.

Model	L1		L2		L3	
	T.A	CR	T.A	CR	T.A	CR
EXAONE-4.0-1.2B	48.1	2.5	43.1	5.8	27.1	0.0
Gemma-3-1B-it	54.9	20.8	55.5	17.8	46.1	13.0
LFM2.5-1.2B	62.3	24.2	54.1	18.7	44.4	9.6
MiniCPM5-1B	79.0	58.3	76.1	44.0	69.7	36.1
Qwen3.5-0.8B	95.2	77.5	90.0	61.3	84.2	51.7
Qwen3.5-2B	98.5	95.0	94.8	85.3	81.8	59.1
G9v3-3B [†]	96.0	83.3	90.7	69.3	81.6	59.6

I Test Suite Coverage Statistics

Table 15 reports the frequency of each complexity feature across the 115 test cases, providing the coverage statistics requested by reviewers. The distribution is skewed toward date-heavy queries (99.1% have nonzero d_{date}), reflecting the importance of temporal reasoning in personal finance. Only 2 of 115 queries contain deliberate misspellings; 22 contain colloquialisms (“can ya”, “tho”, “plz”). Holiday-based queries requiring calendar knowledge are rare ($n = 1$).

Table 16 details the distribution of the 11 major expense categories (Table 7) across the evaluation suite. All 11 classes are represented, led by Food (21.7%) and Transportation (11.3%), while 25.2% of queries test global or unfiltered queries without category constraints.

Table 17 reports per-slot usage frequencies across the 28 `plot_comparison_bars` test cases ($n = 28$, the most parameter-dense tool family). All comparison cases specify baseline and comparison years ($y1, y2$), 85.7% filter by category, 39.3% compare specific single months ($m1/m2$), 21.4% specify multi-month ranges (sm/em), and 10.7% test exact single-day comparisons ($d1/d2$).

Table 15: Complexity feature frequencies across 115 test cases. Each row shows how many test cases have a nonzero value for that complexity feature.

Feature	Count	%
$d_{\text{date}} > 0$ (date specification)	114	99.1
$d_{\text{cat}} > 0$ (category mapping)	37	32.2
$d_{\text{rel}} > 0$ (relative time)	24	20.9
$d_{\text{multi}} > 0$ (compound params)	28	24.3
$d_{\text{abbr}} > 0$ (abbreviated dates)	14	12.2
$d_{\text{hol}} > 0$ (holiday knowledge)	1	0.9
Queries with colloquialisms	22	19.1
Queries with misspellings	2	1.7

Table 16: Major category representation across the 115 test cases mapped to the 11 taxonomy categories in Table 7.

Major Category	Count	%
Food	25	21.7
Transportation	13	11.3
Housing and Utilities	10	8.7
Fitness	7	6.1
Education	7	6.1
Entertainment	6	5.2
Souvenirs/Gifts/Treats	4	3.5
Accommodation	4	3.5
Miscellaneous	4	3.5
Household and Clothing	4	3.5
Electronics and Furniture	2	1.7
Unfiltered / All Categories	29	25.2

Table 17: Parameter slot usage across the 28 `plot_comparison_bars` test cases (Tool 3).

Parameter Slot	Count	%
$y1, y2$ (baseline & comparison years)	28	100.0
<code>category</code> (expense category filter)	24	85.7
$m1, m2$ (single-month comparison)	11	39.3
$sm1, em1, sm2, em2$ (multi-month ranges)	6	21.4
$d1, d2$ (single-day comparison)	3	10.7
$ey1, ey2$ (cross-year range endpoints)	2	7.1
<code>ignore_rent</code> (rent exclusion flag)	2	7.1

J Few-Shot / Test-Set Overlap Analysis

To assess the degree of structural overlap between in-context examples and evaluation queries, we compare each of the 115 test cases against the specialist few-shot examples for the same tool (Table 18).

Table 18: Structural overlap between specialist few-shot examples (73 total across 5 tools) and the 115 test cases. Each test case is compared against the few-shot examples for the same tool.

Overlap Metric	Count (%)
Exact argument slot match	101/115 (87.8%)
Date pattern match	114/115 (99.1%)
High lexical similarity (>0.6)	100/115 (87.0%)
Near-exact match (>0.95)	15/115 (13.0%)

Of the 115 test cases, 101 (87.8%) share the exact same argument slot set as at least one specialist few-shot example for the same tool, and 114 (99.1%) share the same date expression pattern. Lexical similarity (SequenceMatcher ratio) between test queries and their nearest specialist few-shot query exceeds 0.6 for 100 (87.0%) cases, with 15 (13.0%) near-exact matches (>0.95). This structural similarity is expected since both few-shot examples and test cases exercise the same five tools over a fixed parameter schema, but it means that generalization to queries with novel slot combinations or phrasings remains to be validated. Test cases and few-shot examples were authored by the same developer; an independently authored held-out test set would provide a stronger generalization signal.