# DAX-1: Efficient, Executable Spreadsheet Editing

Decide Research Team
Olajide Al-Ameen, Abiodun Adetona
August 2026

**Abstract.** DAX-1 is a specialized system for structured spreadsheet editing. It receives workbook context and a naturallanguage instruction, then returns an executable `spreadsheet_edit_patch_v1` patch. The production configuration combines a quantized Qwen3-14B-derived GGUF model with a project-authored routing and deterministic execution layer. On a frozen internal benchmark containing 350 tasks across seven spreadsheet-editing families, DAX-1 produced 321 strict workbook passes, a 91.7% pass rate. A separate verification run reproduced the same 321/350 result for both verified production constructions. DAX-1 finished seven tasks ahead of the strongest internal checkpoint while reducing the principal model artifact from the 29.5 GB class to 6.79 GB decimal, or 6.32 GiB. Appendix A separates the production system from its standalone and non-quantized variants. The serving result was measured separately on a 70-task production serving gate. DAX-1 recorded 1.48 seconds p50 latency, 2.42 seconds p95 latency, and a maximum measured route footprint of 7,613 MiB. Under a stated warm-GPU assumption of USD 1 per hour, the full 350-task workload costs approximately USD 0.69 in GPU time. The comparison runs supplied for this report cost USD 7.49 for Opus 4.5 and USD 23.90 for Fable 5. Stated narrowly: DAX-1 achieves frontier-level performance on Decide's frozen deterministic spreadsheet-editing benchmark while running at substantially lower estimated serving cost. The benchmark was authored and frozen by Decide, so this is controlled evidence of price-performance rather than an independent public ranking. Section 11 states the limitations in full.

## 1. Introduction

Spreadsheet assistants are often evaluated as chat systems. The user asks a question, the model explains what to do, and the workbook remains unchanged. DAX-1 targets a different job. It must identify the required cells, produce the correct values or formulas, preserve unrelated content, and return a patch that software can apply.

This distinction matters in finance operations, reporting, reconciliation, CRM cleanup, KPI maintenance, and internal business workflows. A plausible explanation has little value when the wrong row is restored or the formula lands in the wrong sheet. The output must be executable, and the change must be verifiable.

DAX-1 focuses on seven task families:

1. Formula repair
2. Lookup and join completion
3. Duplicate removal
4. Formatting cleanup
5. Row deletion and cleanup
6. Date, filter, and sort repair
7. Aggregation and classification repair

The current system does not claim coverage of pivot tables, chart creation, macros, arbitrary financial modeling, cross-workbook imports, or every spreadsheet operation.

## 2. Task and Production Definition

The public name **DAX-1** refers to the quantized production system. It includes the quantized GGUF model, its spreadsheet-specific learned behavior, the patch grammar, and the routing layer used in deployment.

**Evaluation configuration.** DAX-1 is evaluated in its production configuration, using the quantized GGUF model and routing layer.

The routing layer is part of the computation rather than a cosmetic dispatcher. It performs deterministic execution for aggregation, duplicate removal, and row deletion. Formula repair, lookup and join, formatting cleanup, and date/filter/sort repair use the quantized neural path. This design gives exact table operations a controlled implementation while keeping language-sensitive repair behavior in the model.

Names are used consistently throughout. **DAX-1** refers to the production system evaluated in the main result: the quantized model, the patch-generation interface, the deterministic execution path, and the workbook verifier. **DAX-1 Base** is the internal non-quantized checkpoint and appears only in the ablations. Standalone model variants are reported separately in Appendix A and are not the product result.

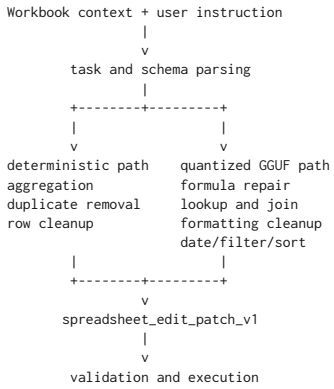
### 2.1 Output contract

DAX-1 returns a versioned JSON patch:

```
{
  "patch_version": "spreadsheet_edit_patch_v1",
  "operations": [
    {
      "op": "set_cell",
      "sheet": "Invoice Computation",
      "cell": "D18",
      "formula": "=B18*C18",
      "number_format": "$#,##0.00"
    }
  ]
}
```

The main operations are `set_cell` for sparse edits and `set_range_values` for row or table repairs. The patch interface keeps generation separate from workbook execution. It also makes failures inspectable: a reviewer can see the target sheet, range, value, formula, and number format before the workbook is changed.

### 2.2 System path




## 3. Dataset, Post-Training, and Recovery

### 3.1 Dataset and executable labels

The training records were built from broken workbook states, repair instructions, and target repaired states. Relevant workbook context was serialized into JSONL prompts, including sheet names, ranges, values, formulas, and formatting information when the task required it. The target was an executable patch rather than a paragraph explanation.

This choice made the training objective match deployment. Each example taught the model to identify the smallest correct edit, preserve unrelated cells, and choose between sparse `set_cell` operations and compact `set_range_values` repairs. The training data covered the same seven operation families used for development evaluation, while the frozen 350-task benchmark remained evaluation-only.

### 3.2 Behavior-first compression

The starting point was a Qwen3-14B-derived spreadsheet checkpoint [1]. The merged full-precision artifact was in the 29.5 GB class. That footprint made low-cost deployment difficult, so the first engineering goal was behavioral retention under aggressive compression.

Direct compression near 4 GB failed the retention gate. The file was attractive and the spreadsheet behavior was not. The decision rule came from the earlier Behavior Before Perplexity work: promote a compact artifact only when its deployed behavior survives, rather than treating perplexity, reconstruction loss, or size as sufficient evidence [4]. In DAX-1, that principle became paired parent-correct retention tests and family-level workbook gates.

An intermediate behavior-first mixed-precision experiment moved the artifact toward 6 GB and retained 20 of the 21 paired parent-correct cases on the original quantization gate. That experiment established the practical size range; it is not the released configuration. The shipped artifact is quantized uniformly to IQ3_S, as recorded in Section 8 and the reproducibility record.

### 3.3 Post-training interference

LoRA recovery then exposed interference between task families [2]. Adapters that improved lookup or row selection could damage formula or formatting behavior. A focused formula-and-lookup adapter reached 10/10 on both target families and reduced formatting to 0/10. Family gains could not be added together by assuming that one adapter would preserve every existing behavior. Recovery candidates therefore had to replay the behaviors already considered solved.

### 3.4 Output and routing recovery

The output layer required its own repair. A 45-case user-style stress test found malformed compound JSON even when the intended operations were recognizable. Grammar-constrained decoding raised valid and schema-conformant output from 75% to 100% with no recorded loops and essentially unchanged latency. The grammar controlled syntax. Semantic correctness remained the responsibility of the model, router, and verifier [3].

The deterministic routes also failed a targeted overfitting audit. They passed unseen values in the known schema and failed every renamed-sheet case because destination names were hardcoded. We changed the parser to recover source and destination sheet names from each prompt. The repaired routes passed 30/30 unseen-value cases, 30/30 renamed-sheet cases, and the original 70/70 gate again.

### 3.5 From retained behavior to 321/350

The final score was not produced by one training run. Behavioral gates first identified a quantization range that retained the parent model's useful spreadsheet behavior. Targeted post-training recovered weak neural families without accepting regressions elsewhere. Grammar decoding made the patch interface reliable. Deterministic execution then covered exact long-table operations that remained unreliable under the bounded neural runs. The renamed-sheet audit removed the last hardcoded routing assumption found before release.

The 321/350 result belongs to that complete production path. Quantization changed the serving envelope, post-training recovered learned behavior, and the execution layer made exact operations dependable enough for deployment.

For context, stock Qwen3-14B achieved 34/350 strict workbook passes, or 9.7%, under the same frozen benchmark and verifier, confirming that the spreadsheet behavior measured here is not present in the base model out of the box. Appendix A gives the full ablation table.

## 4. Evaluation

### 4.1 Frozen benchmark

The benchmark, prompts, verifier, and baseline harness were authored by Decide. The benchmark was frozen before the final DAX-1 production evaluation and held out from training. This makes the result internally controlled but not yet an independent third-party benchmark.

The main benchmark contains 350 internally held tasks, divided evenly across the seven families. Each record contains a workbook context, an instruction, and a required patch representing the expected workbook-changing operations. The frozen evaluation file has SHA-256 `31d5c5eaef4d6efc96e6632c1bba5a188c6c0e707cf256e45aeb95638c6be6bd`.

The evaluation file was kept separate from training. Earlier synthetic data used related task structures, while the frozen 350 records remained evaluation-only.

### 4.2 Strict workbook pass condition

Each generated response is parsed as `spreadsheet_edit_patch_v1`. A valid JSON response receives no credit by itself. The patch is evaluated against the required workbook-changing operations, including sheet names, cells or ranges, values, formulas, and number formats where applicable. A strict pass is counted only when the resulting workbook state matches the expected repair under the verifier.

Independent operations are evaluated as an unordered set. Spreadsheet edits that do not depend on one another have no meaningful generation order. The published 321/350 score is the locked production result from the final verification run.

### 4.3 Verifier

The verifier applies each generated patch to the corrupted workbook and saves the resulting workbook. It then compares the output workbook against the target repaired workbook cell by cell.

A value pass requires matching workbook values after normalization. Blank cells are normalized to `None`; floats are rounded and compared with a 1e-9 tolerance; datetimes, dates, times, and timedeltas are normalized to comparable serialized forms. A strict pass requires a value pass plus exact number-format string matches for nonblank cells.

Formulas are compared as workbook cell contents, not as algebraically equivalent expressions. If the target contains `=B18+C18`, an output of `=SUM(B18:C18)` does not receive credit unless it matches the target workbook representation the verifier expects. This scoring is intentionally strict because production spreadsheet workflows require the final workbook state to match the expected repair, not merely a semantically plausible answer.

### 4.4 Production routing split

The benchmark contains 150 tasks from the three deterministic families and 200 tasks from the four neural families.

| Execution path | Families | Tasks |
|---|---|---|
| Deterministic execution | Aggregation, duplicate removal, row deletion | 150 |
| Quantized GGUF | Date/filter/sort, formatting, formula, lookup/join | 200 |
| **Total** | **Seven families** | **350** |

This split is part of the reported system. The 321/350 score should always be read as a production-system result.

### 4.5 Inference and latency protocol

All main baselines were evaluated under a one-shot patch-generation protocol. Each model received one attempt per task. Failed parses, invalid patches, and incorrect workbook states counted as failures. No model received verifier feedback or retry-based repair in the main comparison.

Model generation used temperature 0, thinking disabled, one attempt, and an 8,192-token context. The 1.48-second p50 and 2.42-second p95 latency figures come from the separate 70-task production serving gate. That gate covered ten tasks per family and recorded zero runtime errors.

The main result is a single frozen benchmark run per system unless otherwise stated. Because model APIs and sampling behavior can vary, future versions of this report will include repeated runs and paired statistical tests.

Latency comparisons use saved profiles from the same spreadsheet patch-generation protocol. Opus 4.6 is excluded from the main latency figure because its supplied latency estimate came from only two local smoke tasks.

## 5. Main Result

The final verification run produced the following result:

| System | Configuration | Strict passes | Pass rate |
|---|---|---|---|
| **DAX-1** | **Quantized production system** | **321/350** | **91.7%** |
| GPT-5.5 | Frontier API baseline | 319/350 | 91.1% |
| Fable 5 | Frontier API baseline | 204/350 | 58.3% |
| Opus 4.5 | Frontier API baseline | 186/350 | 53.1% |

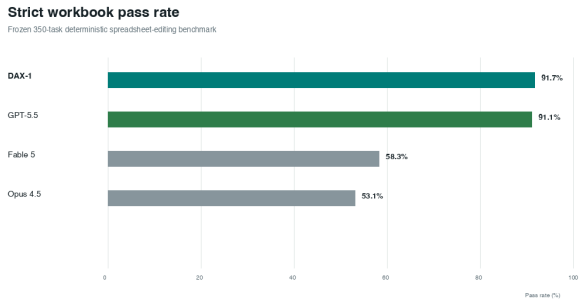



Figure 1. Strict workbook pass rate on the frozen 350-task deterministic spreadsheet-editing benchmark, main baselines only. DAX-1 is the quantized production system. The provisional Opus 4.6 run is recorded in Appendix B.

The main table lists only baselines whose run conditions are documented and comparable. A provisional run is recorded separately in Appendix B.

DAX-1 finished two tasks ahead of GPT-5.5 in the measured run. The difference is small enough that the report treats the two systems as close on accuracy. The larger separation appears in serving cost and latency.

DAX-1 also finished seven tasks ahead of the strongest internal checkpoint. The gain belongs to the production configuration as a whole, including the routing layer. Appendix A records the ablations.

### 5.1 Family results

| Family | Passes | Pass rate |
|---|---|---|
| Aggregation and classification | 42/50 | 84% |
| Date, filter, and sort | 40/50 | 80% |
| Duplicate removal | 50/50 | 100% |
| Formatting cleanup | 50/50 | 100% |
| Formula repair | 50/50 | 100% |
| Lookup and join | 40/50 | 80% |
| Row deletion and cleanup | 49/50 | 98% |
| **Overall** | **321/350** | **91.7%** |

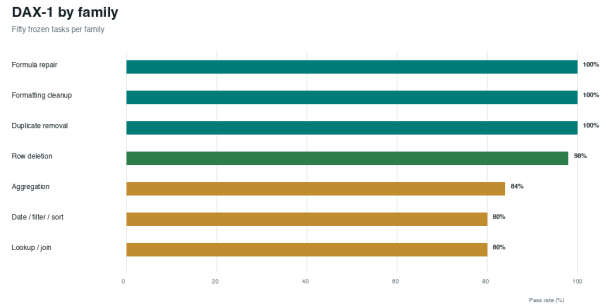



Figure 2. DAX-1 family-level pass rate. The benchmark contains 50 tasks in each family.

The aggregate score hides a clear shape. Duplicate removal, formatting cleanup, and formula repair were perfect on this set. Row deletion missed one supplied target. Aggregation, date/filter/sort, and lookup/join account for 28 of the 29 recorded failures; the remaining failure is the row-deletion miss noted above.

**Error analysis.** Two of the three weak families run on the quantized model path, where the failures are generation failures. Aggregation is the case that needs explaining, because it is routed to deterministic execution and still records eight misses. Deterministic execution cannot produce an arithmetic error, so those failures occur before it: the route has to identify the target range, the destination cell, and the label from the instruction, and hand them to the executor. A misread range or destination produces a well-formed patch that writes the right kind of edit in the wrong place, which the verifier scores as a miss. The deterministic families therefore inherit the model's parsing accuracy even though their execution is exact, and aggregation carries the most parsing surface of the three.

The result also identifies where additional work has value. More training on already perfect families would add benchmark risk with little upside. Future recovery should concentrate on aggregation localization, date normalization and selection, and lookup range alignment.

# 6. Latency and Serving Footprint

The quantized production model recorded a maximum route footprint of 7,613 MiB at an 8,192-token context. The principal GGUF artifact contains the spreadsheet adapter and uses the `MOSTLY_IQ3_S` file type.

| System | p50 latency | p95 latency | Evidence scope |
|---|---|---|---|
| **DAX-1** | **1.48 s** | **2.42 s** | 70-task production serving gate |
| Opus 4.5 | 2.64 s | 6.54 s | Saved comparison profile |
| Fable 5 | 7.71 s | 15.85 s | Saved comparison profile |
| GPT-5.5 | 7.84 s | 20.50 s | Saved comparison profile |

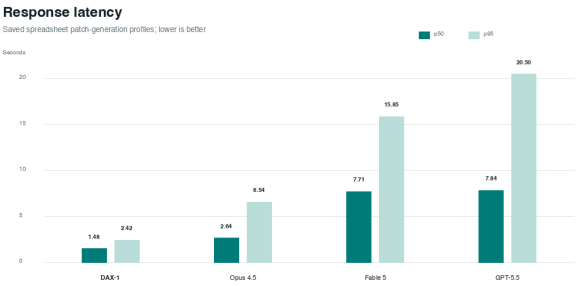



Figure 3. Measured p50 and p95 latency under the spreadsheet patch-generation protocol. Lower is better.

The DAX-1 median was 5.3 times faster than the GPT-5.5 median in these profiles. Its p95 was 8.5 times faster. These numbers describe the measured benchmark configuration. Production latency will still depend on GPU choice, batching, prompt length, queueing, and server utilization.

The artifact itself is 6,788,274,816 bytes. This is 6.79 GB in decimal units and 6.32 GiB in binary units. Compared with the 29.5 GB-class full-precision artifact, the principal model file is approximately 77% smaller.

# 7. Cost

The DAX-1 cost figure is a serving estimate based on measured runtime and a warm GPU priced at USD 1 per hour. The workload consumed approximately 0.6856 GPU-hours:

```
350-task DAX-1 cost = 0.6856 GPU-hours * USD 1.00/hour
                    = USD 0.69
```

The comparison costs supplied for Opus 4.5 and Fable 5 are measured API-run costs.

The cost estimate and the latency table are not the same measurement and should not be reconciled against each other. The USD 0.69 figure covers full benchmark wall-clock serving time, including model warmup, request orchestration, and non-generation overhead; dividing it across 350 tasks implies roughly 7.05 seconds per task. The latency table reports per-task generation latency from a separate 70-task production serving gate, where the median was 1.48 seconds. The two numbers measure different things and are not interchangeable.

| System | 350-task cost | Passes | Cost per verified pass |
|---|---|---|---|
| **DAX-1** | **USD 0.69** | **321** | **USD 0.0021** |
| Opus 4.5 | USD 7.49 | 186 | USD 0.0403 |
| Fable 5 | USD 23.90 | 204 | USD 0.1172 |

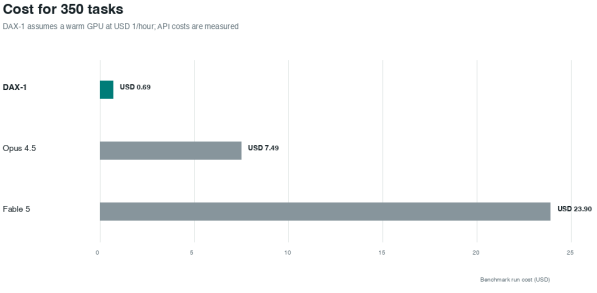



Figure 4. Cost of the 350-task run. DAX-1 uses a USD 1/hour warm-GPU assumption; API baselines use measured run costs.

Under that assumption, DAX-1 costs 90.8% less than Opus 4.5 and 97.1% less than Fable 5 for the full benchmark. The estimate excludes idle capacity, storage, orchestration,

engineering labor, and network costs. A real hosted service must account for utilization. The figure remains useful because it makes the assumption explicit and can be recalculated for any effective hourly rate.

## 8. Quantization and Recovery

The compression result matters because it changed the hardware envelope. A 29.5 GB-class checkpoint requires a different serving plan from a 6.79 GB artifact with a measured 7,613 MiB route footprint.

The winning path was:

1. Post-train the Qwen3-14B-derived model for structured spreadsheet patches.
2. Merge the selected spreadsheet adapter into the parent model.
3. Convert the merged model to F16 GGUF.
4. Quantize uniformly to IQ3_S.
5. Use grammar-constrained generation for the patch schema.
6. Route three exact table families through deterministic execution.
7. Evaluate the complete production path.

Two production constructions reached the locked 321/350 result: the final GGUF plus router, and the IQ3_S plus Giant LoRA plus router. Both figures are aggregate scores. Per-task agreement between the two constructions has not been reported here, so this should be read as two constructions reaching the same total, not as a demonstration that they pass and fail the same tasks. They share one public identity because users experience the production system rather than the research artifact layout.

Several earlier candidates were smaller. They were rejected after behavioral evaluation. File size was treated as a deployment constraint, never as evidence of spreadsheet quality. That decision prevented a compact but materially weaker artifact from becoming the release.

## 9. Production Routing Audits

Routing created a second responsibility: the execution layer had to generalize beyond the exact wording and sheet names used during development.

The first post-freeze audit generated a new 70-row split with no duplicate IDs, prompts, or conversation hashes. The deterministic families passed 30/30 under the known schema. A second audit changed source and destination sheet names while preserving task semantics. The existing router failed 0/30 because destination names had been hardcoded.

The repair parsed sheet names from each prompt and failed closed when required names were absent. After the change:

| Audit | Result |
|---|---|
| Original deterministic subset | 30/30 |
| Post-freeze unseen values | 30/30 |
| Renamed source and destination sheets | 30/30 |
| Original semantic gate | 70/70 |

This audit improved the production system and narrowed the claim. It demonstrated invariance to the tested sheet-name changes. It did not test arbitrary schemas, arbitrary user language, or unimplemented operations.

## 10. What the Result Supports

The evidence supports these statements:

- DAX-1 is a quantized production system for seven structured spreadsheet-editing families.
- It passed 321/350 tasks, or 91.7%, on the frozen internal benchmark.
- A separate verification run reproduced 321/350 for two production configurations.
- It finished seven tasks ahead of the strongest internal checkpoint.
- Stock Qwen3-14B achieved 34/350 strict workbook passes, or 9.7%, under the same frozen benchmark and verifier.
- Its principal GGUF artifact is 6.79 GB decimal, or 6.32 GiB.
- The 70-task serving gate measured 1.48 seconds p50, 2.42 seconds p95, zero runtime errors, and at most 7,613 MiB across the neural routes.
- Under a USD 1/hour warm-GPU assumption, the measured benchmark runtime corresponds to approximately USD 0.69.

One boundary belongs here rather than in Section 11, because it concerns what the main number attributes: the 321/350 is a production-system result, and it does not show that the quantized GGUF alone produced it. Appendix A separates the contributions. The scope of the benchmark and what it does not test are stated once, in Section 11.

## 11. Limitations

DAX-1 is evaluated on an internal frozen benchmark designed around deterministic spreadsheet-editing tasks. The benchmark is held out from training and measures whether a generated patch produces the expected final workbook state. This gives us a controlled way to measure workbook-editing performance, but it is not a substitute for broad external evaluation.

The current results should be interpreted within the task scope of the benchmark: formula repair, lookup and join completion, duplicate cleanup, formatting cleanup, row cleanup, date/filter/sort repair, and aggregation repair. DAX-1 is not intended to represent general spreadsheet intelligence, and the benchmark does not test macros, pivot tables, chart creation, broad financial modeling, or open-ended workbook analysis.

For stronger public reproducibility, we plan to release representative tasks, the verifier code, and an external evaluation endpoint. We will not release raw frozen benchmark answers, and they will remain excluded from training. This keeps the benchmark useful as a measurement tool while allowing external users to understand and test the evaluation protocol.

# 12. Related Work

**Spreadsheet agents and spreadsheet representation.** Prior systems treat spreadsheets mainly as an agent-control problem. SheetCopilot introduced atomic spreadsheet actions driven by a planner [5]. SheetAgent separates planning, retrieval, and manipulation across collaborating modules and evaluates on long-horizon, reasoning-dependent tasks [6]. SheetMind frames the same job as an end-to-end multi-agent pipeline [7]. SpreadsheetLLM attacks the problem from representation instead, compressing workbook structure so a model can read tabular layout within a token budget [8]. DAX-1 differs in target rather than ambition: it does not plan multi-step sessions or hold a conversation about a workbook. It emits one executable patch for one instruction and is scored on whether the resulting workbook matches the expected repair.

**Constrained decoding and structured output.** The patch interface depends on constrained generation rather than prompt discipline. PICARD showed that incremental parsing during decoding turns a fine-tuned model into a reliable producer of a formal language [9]. Grammar-constrained decoding generalized this to structured NLP tasks without finetuning [10], and finite-state approaches made guided generation efficient enough to be routine [11]. DAX-1 uses GBNF grammars in llama.cpp [3] for the same reason: the patch schema becomes a guarantee instead of a convention.

**Quantization.** The compression path sits alongside post-training quantization methods such as GPTQ [12] and AWQ [13], which recover accuracy at low bit widths through calibration and activation-aware scaling. The distinguishing choice here is the promotion rule rather than the algorithm. Dutta et al. show that matched benchmark accuracy can hide substantial behavioral change after compression, with answers flipping in both directions [14]; that failure mode is exactly what the paired parent-correct gates in Section 3.2 are built to catch, and it is the argument behind promoting on retained deployed behavior rather than perplexity or reconstruction loss [4].

**Adapters and merging.** Adapter-based post-training carries known trade-offs. LoRA learns less than full finetuning while forgetting less of the base model [16], which matches the family-interference behavior recorded in Section 3.3, where an adapter that fixed lookup and row selection degraded formatting. TIES-Merging addresses the interference that appears when task-specific models are combined [15], the general form of the problem encountered when merging the spreadsheet adapter into the parent model.

**Evaluation and contamination.** The benchmark here is authored internally and frozen, which places it squarely in the space that the data-contamination literature warns about; Section 4.1 states the ownership rather than leaving the reader to ask. The main result is also a single run per system. Paired designs over per-task outcomes, tested with McNemar's test [17], are the appropriate next step and are planned for a future revision.

# 13. Reproducibility Record

| Item | Recorded value |
|---|---|
| Production artifact | `dax1-final.gguf` |
| Artifact size | 6,788,274,816 bytes |
| Artifact SHA-256 | `6cbb881a03aae833bd1e044b4cfc5d450376ab734f6c594361ee9a1a39d0a9c5` |
| GGUF file type | `MOSTLY_IQ3_S` |
| Adapter state | Baked into principal GGUF |
| Frozen benchmark | 350 tasks, seven families |
| Benchmark SHA-256 | `31d5c5eaef4d6efc96e6632c1bba5a188c6c0e707cf256e45aeb95638c6be6bd` |
| Public result | 321/350, 91.7% |
| Valid JSON | 350/350 |
| Context | 8,192 tokens |
| Generation | Temperature 0, thinking disabled, one attempt |

The report uses the locked 321/350 verification run as the public result. Internal development receipts contain later scorer experiments and alternative recompositions. Those experiments do not replace the production result reported here.

# 14. Conclusion

DAX-1 began as a deployment problem. The full-precision spreadsheet checkpoint was capable and too large for the serving target. Direct low-bit compression damaged the behavior that mattered. The work moved through behavioral retention tests, post-training, grammar hardening, deterministic execution, and targeted routing audits.

The final production system passed 321 of 350 frozen tasks. It finished seven tasks ahead of the strongest internal checkpoint and two tasks ahead of GPT-5.5 in the supplied comparison run. Its principal artifact fits in 6.79 GB, and the measured serving gate stayed below 7,613 MiB with a 1.48-second median response.

That is the useful result. On Decide's frozen one-shot verified spreadsheet-editing benchmark, DAX-1 achieved 321/350 strict workbook passes, narrowly ahead of GPT-5.5 and substantially ahead of Fable 5 and Opus 4.5, at much lower estimated serving cost. The benchmark is internal, so the result is controlled evidence of price-performance rather than a final independent public ranking.

# Appendix A. Ablations

The main result is a production-system number. This table separates what the model contributes from what quantization and the production layer contribute.

| System variant | Strict passes | Pass rate | Purpose |
|---|---|---|---|
| **DAX-1 production system** | **321/350** | **91.7%** | Public production result |
| DAX-1 Base, internal non-quantized checkpoint | 314/350 | 89.7% | Pre-quantized internal baseline |
| DAX-1 standalone GGUF, no production routing | 293/350 | 83.7% | Quantized standalone behavior |
| DAX-1 native standalone candidate | 301/350 | 86.0% | Stronger standalone quantized candidate |
| Stock Qwen3-14B | 34/350 | 9.7% | Base-model capability under the same patch protocol |

On the same frozen 350-task benchmark, stock Qwen3-14B achieved 34/350 strict workbook passes, or 9.7%, compared with 321/350, or 91.7%, for the DAX-1 production system. This closes the base-model ablation gap and shows that the underlying Qwen3-14B model does not solve the task out of the box under the same patch-generation protocol and workbook verifier. The improvement should be attributed to the complete DAX-1 development path, including spreadsheet-specific post-training and production-system design, rather than to the base model alone. The delta bundles those contributions together and does not weigh them individually.

The production system finished seven tasks ahead of the strongest internal checkpoint and twenty-eight ahead of the standalone quantized artifact. That spread should not be read as evidence that quantization improved the model. Routing, deterministic execution, conversion, post-training, and serving controls all contribute.

## Appendix B. Provisional and Stress-Test Baselines

These runs are recorded for completeness and excluded from the main comparison because their run conditions are not yet documented to the same standard.

| System | Strict passes | Pass rate | Caveat |
|---|---|---|---|
| Opus 4.6 | 77/350 | 22.0% | Provisional run. The unexpectedly low score requires further investigation of the model snapshot and prompt handling before it is comparable. |
| Gemini 3.1 Pro Preview | 0/350 | 0.0% | Severe schema failure under our patch-output protocol. |
| Gemini 3.5 Flash | 0/350 | 0.0% | Severe schema failure under our patch-output protocol. |
| DeepSeek V3.2 | 0/350 | 0.0% | Better JSON than Gemini, but no strict workbook passes. |

Gemini and DeepSeek are reported as protocol stress tests. Their low scores reflect failure under our current `spreadsheet_edit_patch_v1` output protocol and should not be read as a general claim about their spreadsheet ability.

The stage at which each run failed is recorded below. A valid JSON rate well above the applied-patch rate means the model emitted parseable output that the executor rejected; an applied-patch rate above the value pass rate means the patch ran but wrote the wrong workbook state.

| Model | Value pass rate | Valid JSON rate | Applied patch rate |
|---|---|---|---|
| Gemini 3.1 Pro Preview | 0.0% | 1.71% | 1.71% |
| Gemini 3.5 Flash | 0.0% | 1.14% | 0.86% |
| DeepSeek V3.2 | 2.0% | 60.29% | 26.57% |

## Appendix C. Evidence Map

| Evidence | Purpose |
|---|---|
| `gguf_350_final.original_receipt.json` | Locked 321/350 production result and family breakdown |
| `routed_gate_receipt.json` | 70/70 gate, latency, route split, and VRAM |
| `overfit_repair_receipt.json` | Unseen-value and renamed-sheet audits |
| `dax1-final.corrected_receipt.json` | Final artifact size, hash, and GGUF type |
| `dax_full350_semantic_eval.jsonl` | Frozen internal 350-task evaluation |
| Researcher handoff, 2026-08-13 | Public framing, baseline comparisons, cost, and release boundaries |

## References

[1] An Yang et al. "Qwen3 Technical Report." arXiv:2505.09388, 2025. https://arxiv.org/abs/2505.09388

[2] Edward J. Hu et al. "LoRA: Low-Rank Adaptation of Large Language Models." arXiv:2106.09685, 2021. https://arxiv.org/abs/2106.09685

[3] ggml-org. "GBNF Guide." llama.cpp documentation. https://github.com/ggml-org/llama.cpp/blob/master/grammars/README.md

[4] Al-ameen. "Behavior Before Perplexity: A Failure-Driven Recipe for Compressing a 27B Agentic Model to 8.39 GB." Bad Theory Labs, 2026. https://www.badtheorylabs.com/papers/behavior-before-perplexity

[5] Hongxin Zhang et al. "SheetCopilot: Bringing Software Productivity to the Next Level through Large Language Models." Advances in Neural Information Processing Systems 36, 2023.

[6] Yibin Chen et al. "SheetAgent: Towards a Generalist Agent for Spreadsheet Reasoning and Manipulation via Large Language Models." arXiv:2403.03636, 2024. https://arxiv.org/abs/2403.03636

[7] "SheetMind: An End-to-End LLM-Powered Multi-Agent Framework for Spreadsheet Automation." arXiv:2506.12339, 2025. https://arxiv.org/abs/2506.12339

[8] Yuzhang Tian et al. "SpreadsheetLLM: Encoding Spreadsheets for Large Language Models." arXiv:2407.09025, 2024. https://arxiv.org/abs/2407.09025

[9] Torsten Scholak et al. "PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models." EMNLP 2021. arXiv:2109.05093. https://arxiv.org/abs/2109.05093

[10] Saibo Geng et al. "Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning." arXiv:2305.13971, 2023. https://arxiv.org/abs/2305.13971

[11] Brandon T. Willard and Remi Louf. "Efficient Guided Generation for Large Language Models." arXiv:2307.09702, 2023. https://arxiv.org/abs/2307.09702

[12] Elias Frantar et al. "GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers." ICLR 2023. arXiv:2210.17323. https://arxiv.org/abs/2210.17323

[13] Ji Lin et al. "AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration." MLSys 2024. arXiv:2306.00978. https://arxiv.org/abs/2306.00978

[14] Abhinav Dutta et al. "Accuracy is Not All You Need." arXiv:2407.09141, 2024. https://arxiv.org/abs/2407.09141

[15] Prateek Yadav et al. "TIES-Merging: Resolving Interference When Merging Models." NeurIPS 2023. arXiv:2306.01708. https://arxiv.org/abs/2306.01708

[16] Dan Biderman et al. "LoRA Learns Less and Forgets Less." Transactions on Machine Learning Research, 2024. arXiv:2405.09673. https://arxiv.org/abs/2405.09673

[17] Quinn McNemar. "Note on the sampling error of the difference between correlated proportions or percentages." Psychometrika 12(2), 1947.