TestGate: Why Evidence Beats Uncertainty for Local LLM Reliability

Rana Muhammad Usman
usmanashrafrana@gmail.com

February 2026

Abstract

Local language models offer privacy and low cost but frequently produce unreliable outputs—malformed JSON, failing code, or invalid SQL. We present **TestGate**, an open-source toolkit providing three reliability strategies as separate modes: (1) *uncertainty-based routing* using token-level entropy and margin from logprobs, (2) *evidence-gated routing* using test execution to decide escalation, and (3) *contract-first generation* using JSON schema validation with repair loops.

We evaluate these strategies on HumanEval (code) and structured output tasks (JSON, SQL). Our key finding is counter-intuitive: **uncertainty-based routing hurts code generation** (pass@1 drops from 0.54 to 0.42), while **evidence-gated routing helps significantly** (pass@1 improves from 0.54 to 0.72, beating even the larger model alone at 0.68). For structured outputs, contract-first generation improves validity dramatically for code-specialized models (0.70 to 1.00) but can hurt general-purpose models.

We provide practical guidelines: use evidence-gating when tests exist, contract-first for structured output with code-specialized models, and avoid uncertainty-based routing for code. TestGate is released as open-source to help practitioners build reliable local LLM applications.

1 Introduction

Local LLMs are increasingly deployed for privacy-sensitive and cost-conscious applications. However, reliability remains a critical barrier: models produce syntactically invalid outputs, code that fails tests, or structured data that doesn't match schemas. Unlike hosted APIs with guaranteed structured output modes, local deployments must handle reliability at the application layer.

We address a practical question: **How should local LLM applications detect and recover from unreliable outputs?** Prior work has explored constrained decoding [12, 8], API-level structured outputs [10], and prompt engineering. We take a different approach: *post-generation reliability strategies* that work with any local model without modifying the inference engine.

We present **TestGate**, a toolkit with three strategies:

1. **Uncertainty-based routing**: Use token-level logprobs to estimate confidence. Escalate to a larger model or repair uncertain spans when entropy is high or margin is low.

2. **Evidence-gated routing**: For tasks with verifiable outputs (code with tests), run the output and only escalate if it fails. This uses ground truth rather than proxy signals.

3. **Contract-first generation**: Request structured JSON matching a schema, validate, repair if invalid, then compile to final artifacts (SQL, code stubs).

Our empirical evaluation yields surprising findings. On HumanEval code generation:

- Uncertainty-based escalation *reduces* pass@1 from 0.54 to 0.52
- Uncertainty-based repair *reduces* pass@1 from 0.54 to 0.42
- Evidence-gated routing *improves* pass@1 from 0.54 to 0.72 (+18 percentage points)

On structured output tasks:

- Contract-first improves SQL validity from 0.00 to 1.00 (code-specialized models)
- Contract-first *reduces* validity from 0.96 to 0.68 (general-purpose models)

These results challenge common assumptions and provide actionable guidelines for practitioners. We release TestGate as open-source at `https://github.com/ranausmanai/testgate`.

2 Related Work

Constrained decoding. Outlines [12] uses finite-state machines to guarantee outputs match JSON schemas. Guidance [8] interleaves generation with programmatic constraints. SGLang [13] compiles structured programs into efficient execution plans. LMQL [2] provides a query language for constrained LLM generation. Grammar-based sampling in llama.cpp [7] restricts tokens via BNF grammars. These require inference engine integration, which may not be available for all deployments.

Uncertainty estimation in LLMs. Token-level entropy and probability margins have been used for selective prediction [11] and calibration. Speculative decoding [5] uses small models to draft tokens verified by larger models. Our uncertainty-based routing extends these ideas to full-output escalation and repair.

Test-driven code generation. Self-repair approaches [9] use test feedback to iteratively fix code. CodeT [3] generates tests alongside code for self-consistency. Our evidence-gating is simpler: run once, escalate on failure, no iteration.

Structured output validation. Instructor [6] validates against Pydantic schemas with retry. Jsonformer [1] guarantees JSON structure by generating only variable portions. Our contract-first approach adds compilation to executable artifacts (SQL, code).

3 The TestGate Toolkit

TestGate provides three reliability strategies as separate CLI modes. Each mode takes a prompt and model configuration, returning output along with metadata about the strategy applied.

3.1 Strategy 1: Uncertainty-Based Routing

We compute per-token uncertainty from logprobs provided by the model backend.

Entropy. For each token, we compute entropy over the top-k logprobs:

$$H_i = -\sum_{j=1}^{k} p_{ij} \log p_{ij}$$

where p_{ij} is the normalized probability of the j-th most likely token at position i.

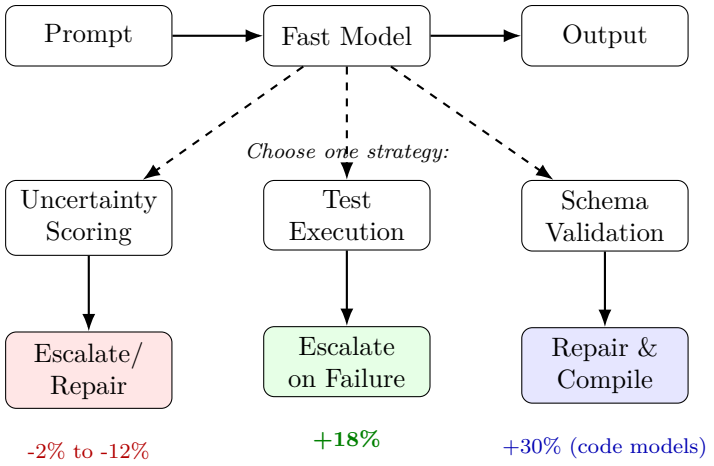

Figure 1: Three reliability strategies with their impact on output quality. Evidence-gating (center) provides the largest improvement for code generation.

Margin. The probability gap between the top two tokens:

$$M_i = p_{i1} - p_{i2}$$

A token is flagged as *uncertain* if $H_i > \tau_H$ or $M_i < \tau_M$ (defaults: $\tau_H = 0.65$, $\tau_M = 0.08$). If more than N tokens are uncertain (default: 4), we trigger escalation or repair.

Escalation. Re-generate with a larger/slower model, optionally passing the draft as context.

Selective Repair. Instead of full re-generation, identify uncertain spans and ask the model to produce targeted JSON patches:

```
{"edits": [{"before": "...", "after": "..."}]}
```

This preserves correct portions of the output while fixing uncertain regions.

3.2 Strategy 2: Evidence-Gated Routing

For tasks with verifiable outputs (code with test cases), we use execution feedback instead of uncertainty estimates.

1. Generate output with the fast model
2. Execute tests on the output
3. If tests pass: return output (no escalation needed)
4. If tests fail: escalate to slow model with error feedback

This approach has a key advantage: it uses *ground truth* rather than proxy signals. A model may be confident but wrong, or uncertain but correct. Tests resolve this ambiguity.

3.3 Strategy 3: Contract-First Generation

For structured outputs (JSON, SQL, code stubs), we decompose generation into steps:

1. **Contract prompt**: Request JSON matching a strict schema
2. **Validation**: Parse and validate against the schema
3. **Repair**: If invalid, request correction (with error message)
4. **Compilation**: Deterministically compile JSON to final artifact

For SQL, the contract is a JSON AST:

```
{"select": ["col1", "col2"], "from": "users",
 "where": [{"col": "age", "op": ">", "val": 18}],
 "order_by": [{"col": "name", "dir": "ASC"}]}
```

which compiles to: `SELECT col1, col2 FROM users WHERE age > 18 ORDER BY name ASC`

This moves syntactic correctness from the model to the compiler.

4 Experimental Setup

4.1 Tasks and Benchmarks

Code generation. HumanEval [4], subset of 50 tasks (the first 50 by task ID). We measure pass@1 with execution-based evaluation. All results are from runs archived in the repository under `eval/runs/`.

Structured output. Custom benchmark with 50 prompts: 20 JSON extraction, 15 SQL generation, 15 Python code stubs. We measure syntactic validity.

4.2 Models

All experiments use local models via Ollama:

- **Fast**: qwen2.5-coder:7b (code-specialized)
- **Slow**: qwen2.5-coder:14b (code-specialized, larger)
- **General**: llama3.2:3b (general-purpose)

4.3 Configurations

Uncertainty routing: entropy threshold 0.65, margin threshold 0.08, min uncertain tokens 4-10.

Evidence-gating: 5-second timeout per test execution.

Contract-first: 1 repair attempt, temperature 0.0.

All experiments run on consumer hardware (Apple M4, 24GB RAM).

Strategy	Pass@1	Gen Time (s)	Slow Used	Change
Baseline (7b only)	0.54	543	0%	—
Baseline (14b only)	0.68	669	100%	+14%
Uncertainty → Escalate	0.52	874	36%	-2%
Uncertainty → Repair	0.42	943	n/a	-12%
Evidence-Gating (7b→14b)	**0.72**	1109	46%	+18%

Table 1: HumanEval results. Uncertainty-based routing hurts; evidence-gating helps.

5 Results

5.1 Code Generation (HumanEval)

Key finding: Uncertainty-based routing *degrades* code generation quality. The model often produces high-entropy outputs that are actually correct, or low-entropy outputs that are wrong. Logprob uncertainty is not a reliable signal for code correctness.

Evidence-gating shows the opposite pattern: by using actual test execution, we escalate only when tests fail (46% of tasks), and the slow model successfully fixes 9 of those 23 failures, boosting pass@1 from 0.54 to 0.72.

5.2 Structured Output

Model	Strategy	Valid Rate	Avg Sec	Repair Rate	Change
Qwen 7B	Baseline	0.70	4.7	0%	—
Qwen 7B	Contract	**1.00**	8.0	6%	+30%
Llama 3B	Baseline	0.96	3.4	0%	—
Llama 3B	Contract	0.68	12.4	36%	-28%

Table 2: Structured output results. Contract-first helps code-specialized models, hurts general models.

Key finding: Contract-first generation is highly effective for code-specialized models but counterproductive for general-purpose models. The SQL contract format (JSON AST) is natural for models trained on code; general models struggle with the added structure.

5.3 Per-Task Analysis

SQL generation shows the starkest contrast: Qwen's baseline produces 0% valid SQL (format confusion, prose mixed with queries), while contract-first achieves 100%. Llama's baseline produces valid SQL directly, but the contract format confuses it.

6 Analysis and Discussion

6.1 Why Does Uncertainty Routing Fail for Code?

We analyzed cases where uncertainty routing made wrong decisions:

Task Type	Model	Baseline	Contract
JSON	Qwen 7B	1.00	1.00
JSON	Llama 3B	1.00	1.00
SQL	Qwen 7B	0.00	**1.00**
SQL	Llama 3B	1.00	0.20
Code Stub	Qwen 7B	1.00	1.00
Code Stub	Llama 3B	0.87	0.73

Table 3: Validity by task type. SQL shows the most dramatic model-dependent behavior.

False positives: Correct code flagged as uncertain. The model may use rare but valid syntax (list comprehensions, lambda expressions) that have high entropy but are correct.

False negatives: Incorrect code with low uncertainty. The model confidently produces plausible-looking code with subtle bugs (off-by-one errors, wrong variable names).

Code correctness is *semantic*, not syntactic. Uncertainty measures syntactic confidence, which is orthogonal to whether the code actually works.

6.2 Why Does Evidence-Gating Work?

Evidence-gating uses the only reliable signal for code: execution. This has limitations (requires tests, only catches failures that tests cover) but avoids the false positive/negative problem entirely.

The pass@1 gain is substantial: 54% to 72% (+18 points). Of 50 tasks, 27 passed immediately on the fast model, and of the 23 that failed and escalated, the slow model fixed 9. Crucially, evidence-gating (72%) even beats using the 14b model alone (68%), while using the slow model for only 46% of tasks. This shows the value of selective escalation: the fast model's correct outputs are preserved, while failures get a second chance.

6.3 Why Does Contract-First Depend on Model Type?

Code-specialized models (Qwen-Coder) are trained on structured formats: JSON, SQL, code. The contract format aligns with their training distribution.

General-purpose models (Llama) may not have seen SQL AST formats. The contract adds cognitive load, leading to more errors than direct SQL generation.

Implication: Match the reliability strategy to the model's strengths.

7 Practical Guidelines

Based on our findings, we recommend:

Do not use uncertainty-based routing for code generation—it hurts more than it helps.

Do use evidence-gating when tests exist; the overhead is minimal and the gains are real.

Do use contract-first for structured output, but only with code-specialized models.

8 Limitations

Benchmark scale. HumanEval has 164 problems; we evaluated 50. Structured output uses a custom 50-prompt benchmark. Larger-scale evaluation would strengthen claims.

Task	Tests Available?	Model Type	Recommended Strategy
Code	Yes	Any	**Evidence-gating**
Code	No	Any	Baseline (no reliable signal)
SQL/JSON	No	Code-specialized	**Contract-first**
SQL/JSON	No	General	Baseline only

Table 4: Decision matrix for choosing reliability strategies.

Model coverage. We tested two model families (Qwen, Llama). Results may differ for other architectures (Mistral, Phi, etc.).

Evidence-gating scope. Requires test cases, which don't exist for all code tasks. Not applicable to non-code domains without oracles.

Uncertainty thresholds. We used default thresholds; task-specific tuning might help. However, the fundamental issue (semantic vs. syntactic confidence) likely remains.

9 Conclusion

We presented TestGate, a toolkit for improving local LLM reliability through adaptive strategies. Our key contributions:

1. **Counter-intuitive finding**: Uncertainty-based routing hurts code generation. Models are often confident when wrong and uncertain when correct.
2. **Evidence-gating works**: Using test execution as the escalation signal improves pass@1 by 18 percentage points (0.54 to 0.72) with zero false positives.
3. **Model-dependent strategies**: Contract-first generation helps code-specialized models dramatically but hurts general-purpose models.
4. **Practical guidelines**: A decision matrix for choosing strategies based on task type, test availability, and model characteristics.

TestGate is released as open-source to help practitioners build reliable local LLM applications. We hope these findings save others from the common assumption that "more uncertainty = escalate" is always beneficial.

References

[1] 1rgs. Jsonformer: A bulletproof way to generate structured json from language models. `https://github.com/1rgs/jsonformer`, 2023. Accessed 2026-02-07.

[2] Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Lmql: A query language for large language models. *arXiv preprint arXiv:2212.06094*, 2022.

[3] Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests. *arXiv preprint arXiv:2207.10397*, 2022.

[4] Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[5] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. *arXiv preprint arXiv:2211.17192*, 2022.

[6] Jason Liu. Instructor: Structured outputs for llms. `https://github.com/jxnl/instructor`, 2023. Accessed 2026-02-07.

[7] llama.cpp contributors. Grammar-based sampling in llama.cpp. `https://github.com/ggerganov/llama.cpp/blob/master/grammars/README.md`, 2023. Accessed 2026-02-07.

[8] Microsoft. Guidance: A cheat code for diffusion models and llms. `https://github.com/guidance-ai/guidance`, 2023. Accessed 2026-02-07.

[9] Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Self-repair: Self-debugging for code generation. *arXiv preprint arXiv:2306.09893*, 2023.

[10] OpenAI. Introducing structured outputs in the api. `https://openai.com/index/introducing-structured-outputs-in-the-api/`, 2024. Accessed 2026-02-07.

[11] Neeraj Varshney, Swaroop Yao, Kaiqi Zhang, and Chitta Baral. Selective prediction for language models. *arXiv preprint arXiv:2210.06770*, 2022.

[12] Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*, 2023.

[13] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, et al. Sglang: Efficient execution of structured language model programs. *arXiv preprint arXiv:2312.07104*, 2023.