ICLR 2026

Rio de Janeiro, Brazil

BeyondBench: Contamination-Resistant Evaluation of Reasoning in Language Models

Gaurav Srivastava[1], Aafiya Hussain[1], Zhenyu Bi[1], Swastik Roy[2], Priya Pitre[1], Meng Lu, Morteza Ziyadi[2], Xuan Wang[1]

[1] Department of Computer Science, Virginia Tech, USA
[2] Amazon AGI, USA

The Problem - Static Benchmarks Are Broken

Why do we need BeyondBench?

- Static benchmarks (GSM8K, MATH, MMLU) are widely used to measure LLM reasoning
- But training data now includes most of the internet - benchmark answers leak into pretraining
- Models score 90%+ on GSM8K, yet when you change the numbers or rephrase the problem, accuracy drops sharply
- We cannot tell if a model is actually reasoning or just recalling memorized answers

The Problem - Static Benchmarks Are Broken

Key Gaps in Existing Benchmarks

1. No mathematical guarantee that solutions are correct and unique
2. No contamination resistance
3. No awareness of model token budgets
4. No support for problems with multiple valid answers

BeyondBench - Our Solution

Algorithmic Problem Generation + Formal Guarantees

Three main ideas:

1. **Generate problems on the fly** – 10^{15} unique instances for each task
2. **Verify every answer** - SAT/CSP solvers confirm unique or fully enumerated solutions
3. **Respect token budgets** - difficulty adapts to model's output window

44 tasks, 117 variations across three difficulty levels

Suite	Tasks	Complexity	Examples
Easy	29	Polynomial	Arithmetic, statistics, counting
Med.	54	Exponential	Fibonacci, primes, geometric sequences
Hard	78	NP-complete	Sudoku, N-Queens, graph coloring, SAT

Scale of Evaluation

101 Models Evaluated - The Largest Study of its Kind

- 85 open-source models (0.5B → 141B parameters)
- 16 proprietary models (GPT-5, Gemini-2.5-pro, o3, etc.)
- 1,000 instances per task for open-source; 100 for proprietary
- Three-fold evaluation for statistical robustness
- All use seed 42, temperature 0.1, top-p 0.9

Main Results - Performance Drops with Complexity

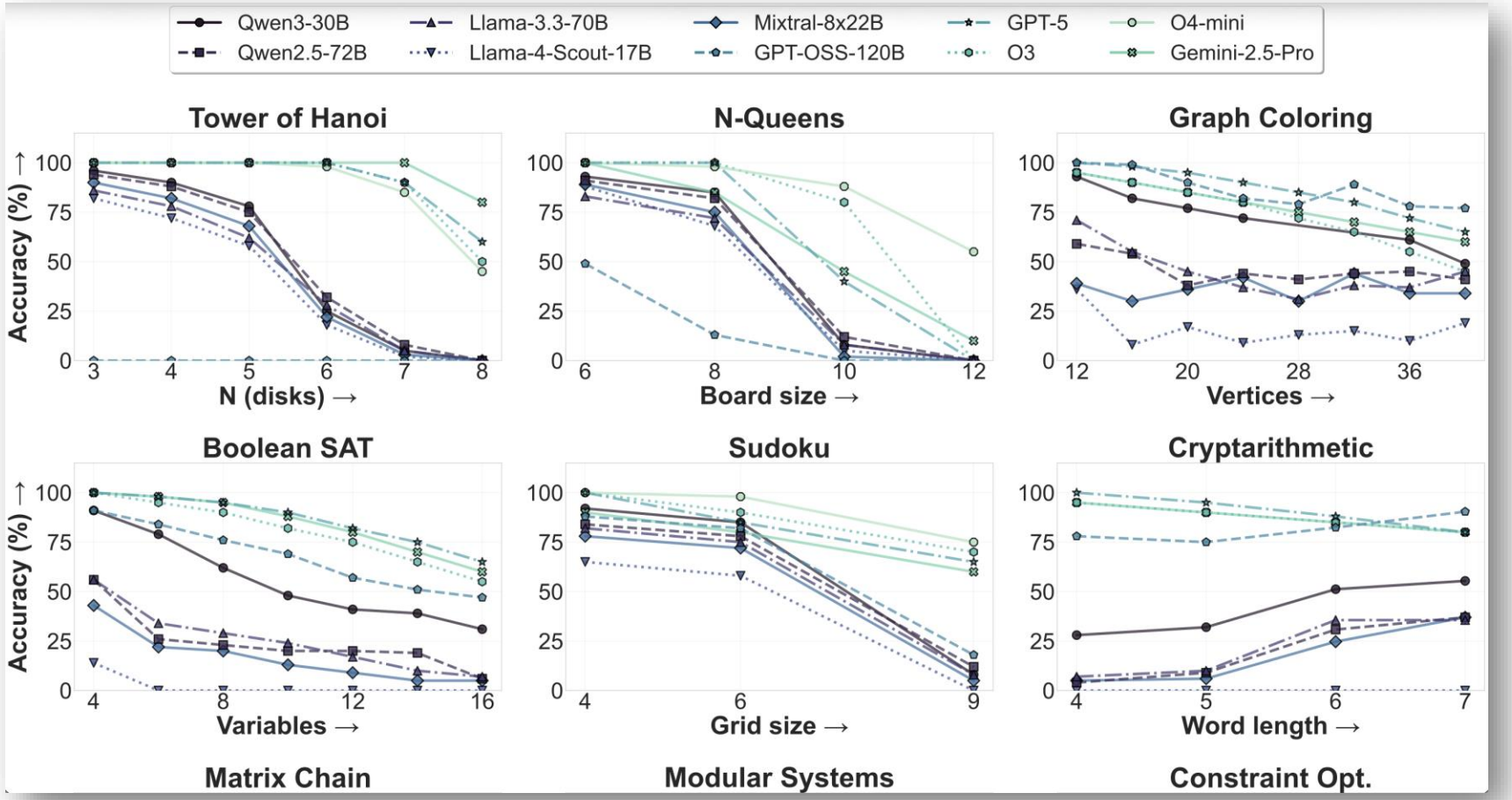
Models Struggle as Problems get Harder

Model	Easy	Med.	Hard	Overall
GPT-5 (tools)	97.3%	81.7%	71.7%	83.6%
GPT-OSS-120B	93.3%	75.3%	59.4%	76.0%
Qwen3-30B-MOE-i	91.9%	73.5%	45.6%	70.3%
Gemini-2.5-pro	89.4%	77.3%	56.2%	74.3%
Llama-3.3-70B	74.8%	46.5%	27.2%	49.5%

- Best open-source model - 93% on Easy but only 59% on Hard
- Most open-source models hit a ceiling around 30-35% on Hard tasks
- Models show sharp performance cliffs (not gradual decline) on NP-complete problems

Main Results - Performance Drops with Complexity

Models Struggle as Problems get harder



Key Findings

Four Surprising Results from our Evaluation

1. **Scaling hits diminishing returns**

   Going from 0.6B to 1.7B gives +17.5%, but 14B to 32B gives only +1.8% (Qwen3 family)

2. **"Thinking" models barely help**

   Qwen3-30B-MOE-thinking improves only 0.2% over the base model

3. **Math fine-tuning hurts**

   Qwen2.5-72B-math scores 48.5% vs 53.4% for the base model (a 4.9% drop!)

4. **Quantization barely matters**

   FP8 and Int8 models lose <3% accuracy; some quantized models even beat full-precision

Tool Use Is the Game Changer

Tool-Augmented Models Outperform Raw Reasoning

Model	With Tools	Without Tools	Drop
GPT-5 (tools)	83.6%	66.8%	-16.8%
GPT-5-mini	81.7%	65.8%	-15.9%
GPT-5-nano	82.0%	38.1%	-43.9%

- Code execution provides the biggest boost (+15-55%)
- Web search gives almost no benefit (+0.3%), confirming contamination resistance
- Smaller models struggle to use tools effectively (41% tool success rate for 3B vs 94% for GPT-5)
- This mirrors human problem-solving: knowing when to compute vs when to reason

Takeaway and Future Directions

What we learned?

- Raw language model reasoning is a real bottleneck - scaling alone will not solve it
- Tool-augmented, agentic models like GPT-5 succeed by recognizing when to compute, not just when to reason
- The path forward: hybrid architectures that combine language understanding with algorithmic tools

BeyondBench is Open-Source and Easy to Use

```
pip install beyondbench
```

Leaderboard : ctrl-gaurav.github.io/BeyondBench

Paper | Code | PyPI : all publicly available!

Acknowledgements and Resources

Supported by

NSF, NAIRR Pilot (PSC Neocortex, NCSA Delta), Cisco Research, NVIDIA, Amazon, Commonwealth Cyber Initiative, Amazon-VT Center for ML, Sanghani Center, VT Innovation Campus

Resources

- Code: github.com/ctrl-gaurav/BeyondBench
- Package: `pip install beyondbench`
- Leaderboard: ctrl-gaurav.github.io/BeyondBench

- Public leaderboard

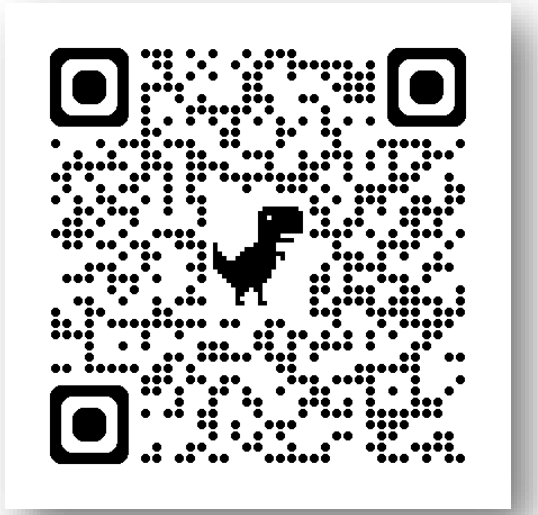
Contact

gks@vt.edu | xuanw@vt.edu

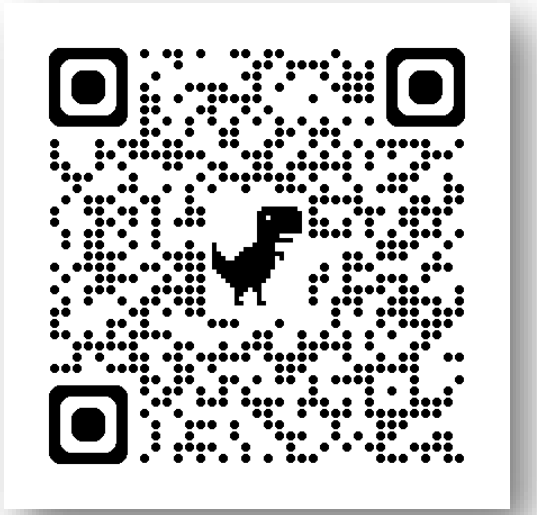
Thanks to

Wang's Lab @ VT & Amazon AGI

↓ Open Review ↓



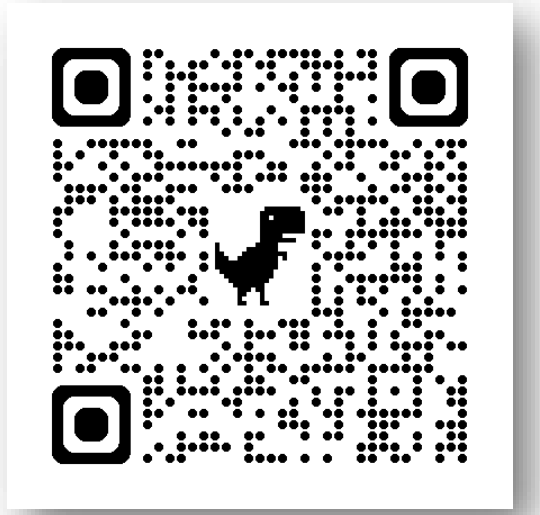
↓ GitHub ↓



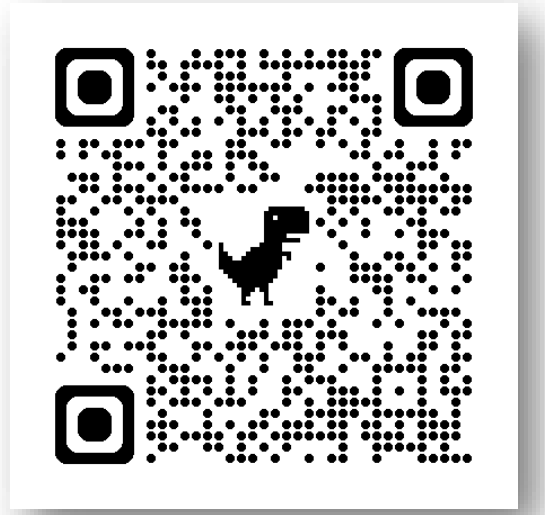
Author Details

Email: gks@vt.edu

↓ Personal Page ↓



↓ LinkedIn ↓



ICLR 2026

Rio de Janeiro, Brazil

Thank You!

ICLR 2026

Rio de Janeiro, Brazil

Appendix

Comparison with Existing Benchmarks

Only BeyondBench Combines All Six Features

Feature	CLRS	DyVal	MathArena	GameArena	NPHard	FCore	PuzzlePlex	BeyondBench
Dynamic generation	✗	✓	✗	✓	✓	✓	✓	✓
Unique solution check	✗	✗	✓	✓	✗	**Partial**	✗	✓
Multi-solution allowed	✗	✗	✗	✗	✗	✗	✗	✓
Scalable difficulty	✓	✓	✗	✗	✓	**Limited**	**Limited**	✓
Contamination-free	✗	✓	✓	✓	✓	✓	✓	✓
Token-aware eval	✗	✗	✗	✗	✗	✗	✗	✓
Number of tasks	30	7	5	3	9	40	15	44

Algorithmic Problem Generation Details

How Problem Generation Works

For each task category, we define a generator: G(parameters, seed) -> problem instance

- Tower of Hanoi: n in [3,12] disks, peg labels from 26 chars, random configs **(>10^18 instances)**
- Sudoku: 30-50 empty cells, digit permutations, cell shuffles **(>10^20 instances)**
- Arithmetic: list length n in [8,64], values in [-1000,1000] **(>10^15 instances)**

Contamination probability

- P(collision) < |training corpus| / |problem space| < 10^12 / 10^15 = 10^-3
- Even if you trained on a trillion examples, chance of seeing an exact instance is <0.1%

Algorithmic Problem Generation Details

Verification

- **Deterministic tasks**: direct computation in O(n)
- **CSP tasks**: solver-based enumeration of all valid solutions
- **Multi-solution tasks**: model gets credit for any valid answer

Token-Aware Evaluation

Calibrating problem difficulty to model token limits

Token requirement for problem p: T(n) = T_prompt(n) + T_solution(n) + T_buffer

- Buffer = 15% of context window for reasoning verbosity
- If T(n) > 0.85 * context window, we scale down: n' = floor(0.8 * n) iteratively

Example - Tower of Hanoi

- n disks need (2^n - 1) * ~12 tokens
- 8 disks = ~3,060 tokens (fits 32K models)
- 10 disks = ~12,276 tokens (needs 128K models)

Token-Aware Evaluation

Post-inference check:

- **VALID**: tokens <= 85% of limit
- **WARNING**: 85-100% of limit (possible truncation)
- **OVERFLOW**: exceeds limit

Scaling Laws Across Model Families

Qwen3 family scaling

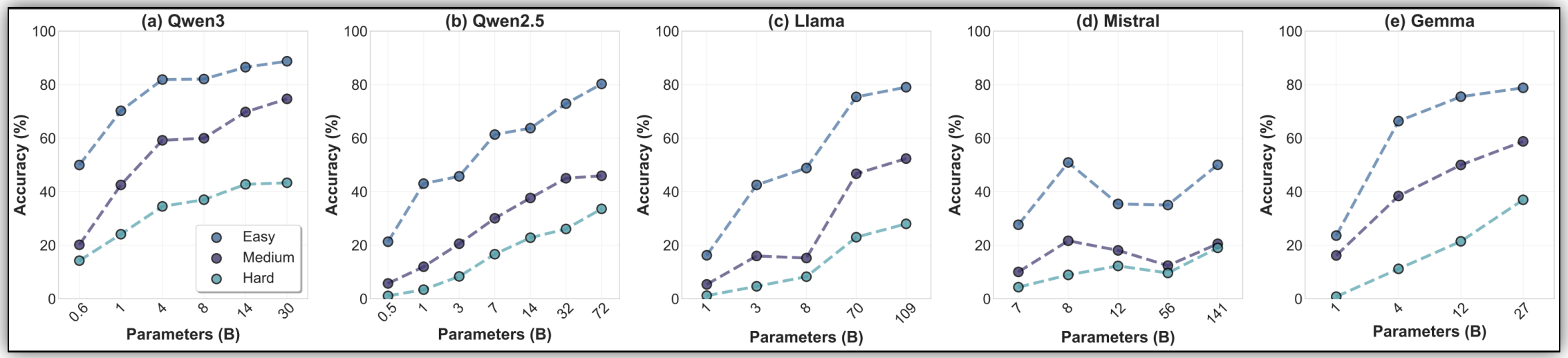
- 0.6B to 1.7B (2.8x params): +17.5% accuracy
- 8B to 14B (1.75x params): +6.5% accuracy
- 14B to 32B (2.3x params): +1.8% accuracy

Qwen2.5 family scaling

- 0.5B to 7B: +26.7% accuracy
- 7B to 72B (10x params): +17.2% accuracy

Scaling Laws Across Model Families

Performance gains follow logarithmic curves



Steep early gains, then diminishing returns!

Performance Collapse Patterns

Two distinct failure patterns

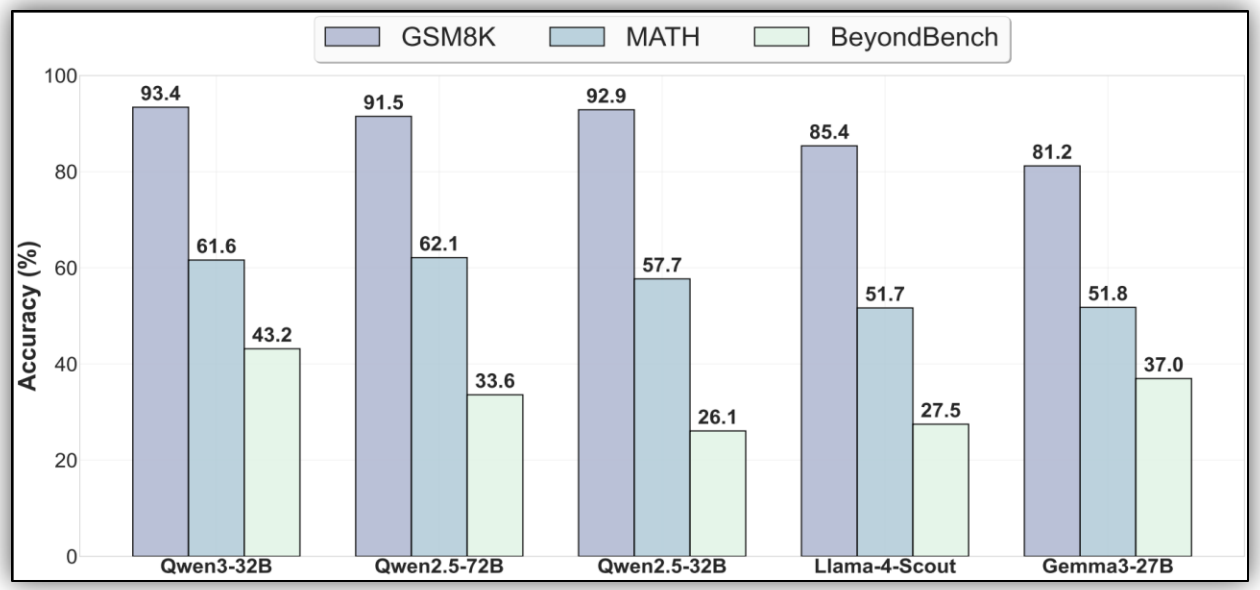
1. **Catastrophic collapse** (Tower of Hanoi, Matrix Chain)
   - Models perform well up to 5-6 disks / 7-10 matrices
   - Then collapse to near-zero performance
   - Requires exponential state management

2. **Counter-intuitive improvement** (Cryptarithmetic, Graph Coloring)
   - Accuracy sometimes increases with problem size
   - Cryptarithmetic: 78% -> 90% for GPT-OSS as word length grows
   - Graph Coloring: relatively stable across 12-40 vertices
   - Richer problem structure may provide more optimization pathways

Even reasoning models (GPT-5, o3) show these same patterns!

Static vs Dynamic Benchmarks

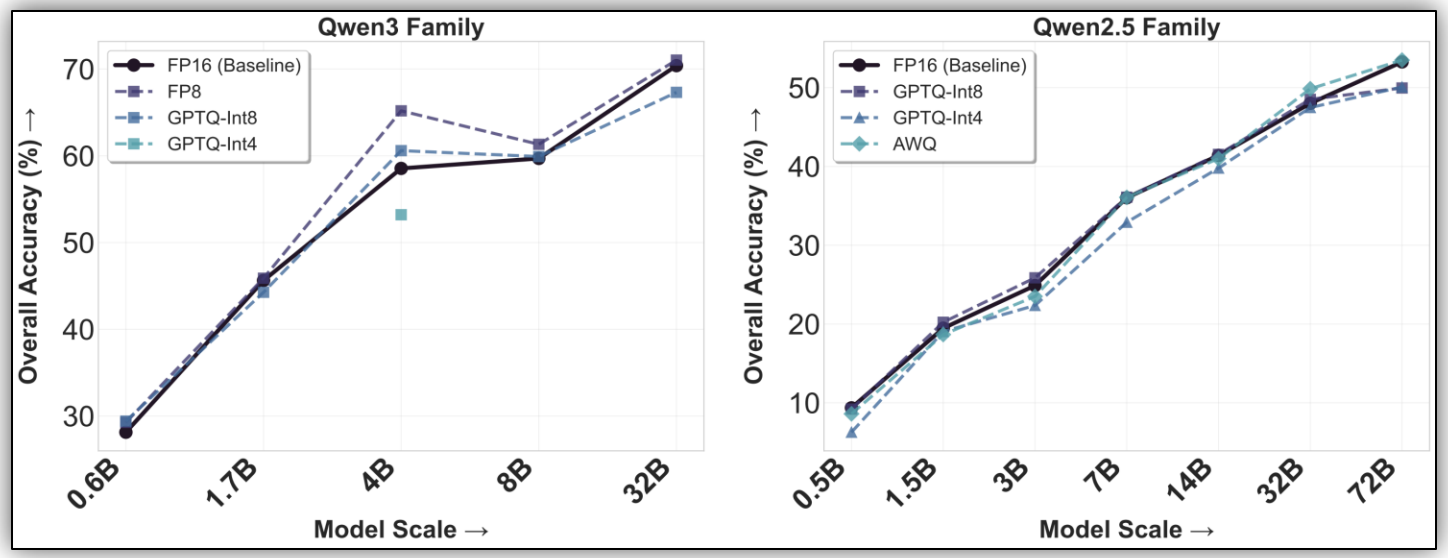
Two distinct failure patterns



- Models scoring 90%+ on static benchmarks (GSM8K) achieve only ~50% on equivalent dynamically generated tasks
- Performance distribution on BeyondBench reflects actual reasoning ability, not memorization
- Confirms that static benchmark scores significantly overestimate model capabilities

Quantization Analysis

Symbolic Operations Over Numerical Precision



- FP8 quantization: nearly identical to full precision
- GPTQ-Int8: <1% drop typically
- GPTQ-Int4 (4x size reduction): only 1-3% drop
- Some quantized variants even outperform full precision (Qwen3-30B-MOE FP8: 71.0% vs 70.3%)

Performance Collapse Patterns

Thinking Models and Reasoning Budget

- Qwen3-4B-thinking: 60.8% vs 58.6% base (+2.1%)
- Qwen3-30B-MOE-thinking: 69.2% vs 69.0% base (+0.2%)

Failure mode analysis

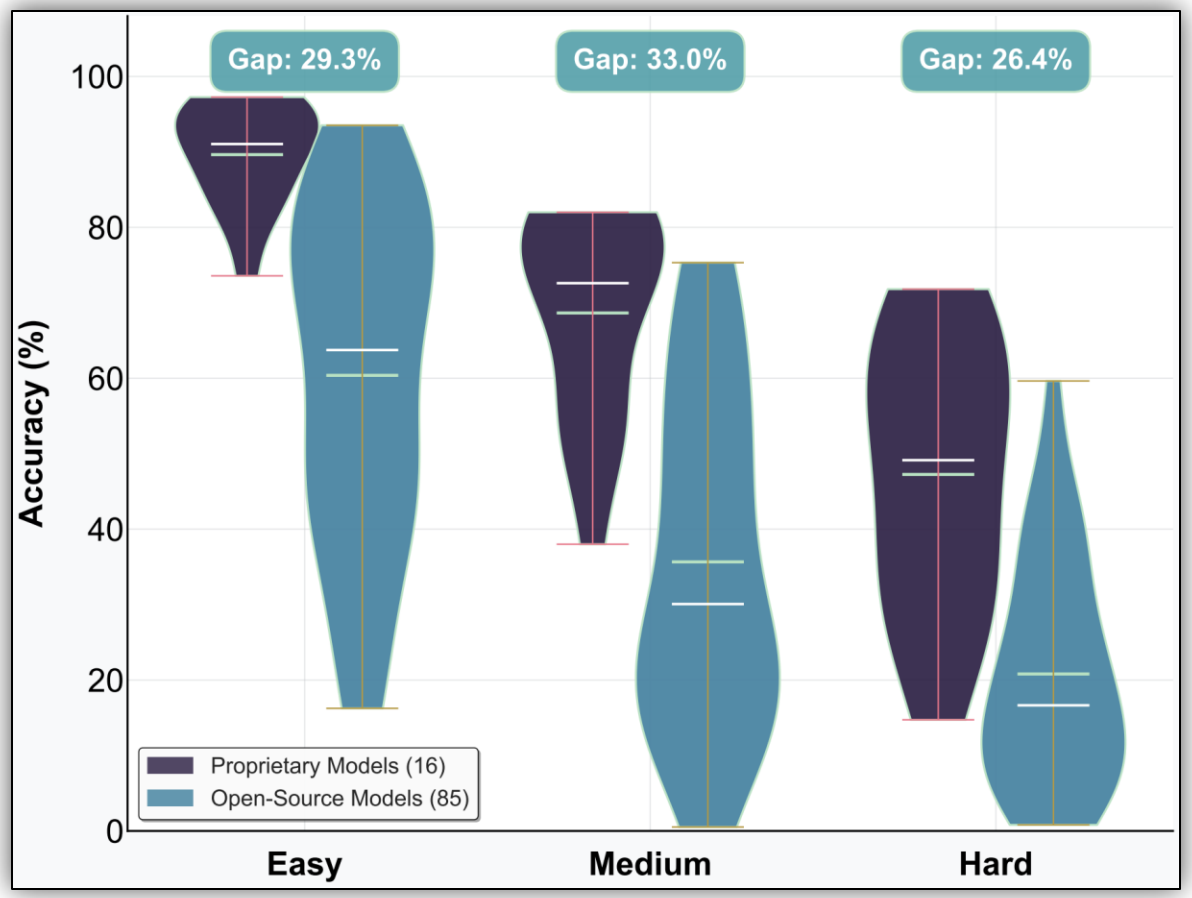
- Reasoning models fail later in execution (step 89/127 vs step 23/127 for vanilla)
- But they struggle with state management during extended reasoning
- Self-correction attempts have 87.6% error introduction rate

Reasoning budget experiments

- Hard tasks benefit most from increased budget (+24-44% for weak models)
- Already-capable models show minimal improvement (o3: <=4% gain)

Performance Collapse Patterns

Performance Distribution



Math Fine-Tuning Hurts

Domain Training Impairs Algorithmic Performance

Model	Base	Math-tuned	Difference
Qwen2.5-72B	53.4%	48.5%	-4.9%
Qwen2.5-7B	36.1%	31.6%	-4.5%

Why?

Mathematical fine-tuning optimizes for symbolic manipulation and equation solving (GSM8K, MATH benchmarks)

Not for algorithmic reasoning tasks that require procedure construction, state management, and systematic search.

Contamination Resistance Validation

SFT and GRPO finetuning on BeyondBench (66k) - 5 models

- Evaluated on different seed (no instance overlap)
- Easy/Medium improvement: +27-38%
- Hard suite improvement: only +10-15%
- NP-complete problems remain fundamentally challenging even after targeted training
- Unlike static benchmarks where training gets near-perfect scores through memorization

Extended Tool Analysis

Per-task analysis across 44 tasks with three tool types

- Code execution: +15-55% improvement (largest gains)
- Calculator: moderate gains on arithmetic tasks
- Web search: +0.3% (nearly nothing - confirms contamination resistance)

Tool orchestration ability varies by model size

- Qwen2.5-3B: 41.2% tool success rate
- GPT-5: 94.2% tool success rate
- Effective tool use itself requires sophisticated reasoning

Open-Source vs Proprietary Gap

Proprietary models likely use internal tool calling

- GPT-5 (proprietary, tool-augmented): 83.6% overall
- GPT-OSS-120B (best open-source): 76.0% overall
- 12.3% gap on Hard tasks alone

The gap cannot be explained by scale alone!

Internal tool calling and code generation, instead of purely on language-based reasoning, helps proprietary models.

Full Leaderboard

Performance of Top Models across Easy, Medium, Hard

Rank	Model	Easy	Medium	Hard	Overall
1	GPT-5 (tools)	97.3%	81.7%	71.7%	83.6%
2	GPT-5-nano (tools)	96.1%	80.1%	69.8%	82.0%
3	GPT-5-mini (tools)	96.1%	79.7%	69.3%	81.7%
4	O3	97.3%	82.3%	61.8%	80.4%
5	o4-mini	94.6%	81.5%	61.4%	79.2%
6	o3-mini	94.2%	80.7%	57.9%	77.6%
7	GPT-OSS-120B	93.3%	75.3%	59.4%	76.0%
8	Gemini-2.5-pro	89.4%	77.3%	56.2%	74.3%
9	GPT-4.1	92.2%	73.1%	46.8%	70.7%
10	Qwen3-30B-MOE-I	91.9%	73.5%	45.6%	70.3%

BeyondBench Pipeline

End-to-End Evaluation Pipeline

Input: Model M, Task suite T, Token limit C

For each task t in T:

1. Select parameters based on token budget C
2. Generate problem instance with random seed
3. Compute full solution set (unique or enumerated)
4. Skip if no valid solution exists
5. Format prompt and run inference
6. Count response tokens, flag warnings if near limit
7. Parse response and check against solution set

Output: Per-task accuracy, instruction-following, token usage

Three guarantees:

- Parameter selection respects token budgets
- Solution set computation covers all valid answers
- Token counting detects when models hit architectural limits

ICLR 2026

Rio de Janeiro, Brazil

Thank You, Once Again!