

Lilavati: Pocket Mathematician

Udgam Labs (udgamlabs.com), October 2026. Technical report for Lilavati and Lilavati-Checker, version 1.0.0, released 16 October 2026.

udgamlabs.com

Abstract

Lilavati is a 94.7M-parameter English language model, trained from scratch by Udgam Labs, that solves grade-school maths word problems by writing a short Python function. The function is checked and run in a restricted sandbox, and its return value is the answer: the model does the reasoning and Python does the arithmetic. A companion model of the same size, Lilavati-Checker, estimates the probability that a program’s result is correct, so that many sampled programs can be combined by checker-weighted voting.

On the GSM8K test set (1,319 problems), Lilavati scores 54.7% with one greedy try and its code executed (55.0% when the released weights are run with the public evaluation code). With 48 tries at temperature 0.7, the checker-weighted vote scores 72.3%, and taking the checker’s single top-rated program (verify48@1) scores 66.7%. Verify48@1 is the protocol used by TinyGSM (Liu et al., 2023), which reports 68.9% for a 125M generator with a 125M verifier. Those weights were not released, and Lilavati is 2.2 points below that reported figure. On the first 300 GSM8K test items, Lilavati’s greedy score (57.3%) is about $2.7\times$ that of the best other open model of up to 150M parameters we measured (21.3%), which answers in text. We also report four robustness sets, a contamination audit, accuracy by the number of reasoning steps, how well the answer’s agreement and confidence signals separate right from wrong answers, speed and memory on an Apple-silicon Mac mini, and the main failure types. The models, the solver package with its sandbox and the evaluation scripts are released under Apache-2.0.

1. Introduction

A small language model given a maths word problem has two jobs. It must work out a plan: which quantities matter and how they combine. And it must carry the plan out without arithmetic slips. A written chain of thought asks the model to do both at once, and at small sizes the arithmetic alone is unreliable.

Program-aided reasoning splits the two jobs (Gao et al., 2022; Chen et al., 2022). The model writes a short program that encodes its plan, and an interpreter computes the result. TinyGSM (Liu et al., 2023) showed that models from 125M parameters upward, trained on millions of synthetic grade-school problems with Python solutions, reach high GSM8K accuracy this way, and that a separate verifier choosing among many sampled programs adds more. Verifiers for maths word problems go back to the paper that introduced GSM8K (Cobbe et al., 2021), and voting over sampled answers to self-consistency (Wang et al., 2022). The TinyGSM models were not released.

Lilavati applies this approach under three constraints:

1. **Small and open.** Lilavati has under 100M parameters and is trained from scratch. Both models have open weights and are small enough to run on a laptop CPU (about 190 MB per model in bf16).
2. **Checkable.** Every answer comes with the program that produced it and with the share of tries that agree.
3. **Safe to run.** Model-written code runs only after a structure check, in a separate, limited process.

Throughout, “code executed” means that the model’s program was run and its return value compared with the reference answer. The other small models we compare against answer in text, their native format, and we say so next to every comparison.

2. Model overview

The name comes from *Līlāvātī* (लीलावती), the book of arithmetic word problems that the mathematician Bhāskara II wrote in 1150 CE.

	Lilavati	Lilavati-Checker
What it does	Reads an English word problem and writes a Python function whose return value is the answer	Reads a problem, one Lilavati program and that program's result, and estimates the probability that the result is correct
Parameters	94.7M	94.7M (same architecture and tokenizer as Lilavati)
Output	A Python function, closed by a code fence	P(correct), from the next-token logits of “yes” and “no”
Hugging Face	UdgamLabs/Lilavati	UdgamLabs/Lilavati-Checker
Licence	Apache-2.0	Apache-2.0

Each Hugging Face repository holds the weights (model.safetensors, bf16), config.json, generation_config.json and the tokenizer files. The GitHub repository [UdgamLabs/lilavati](#) holds the lilavati Python package, a command-line tool, a local Gradio playground, the evaluation scripts and SHA-256 checksums of the released weight files.

Interface. The package wraps program writing, safe execution and voting in one call:

```
from lilavati import Lilavati
lv = Lilavati() # downloads both models; picks cuda, mps or cpu
r = lv.solve("A shop sells pencils at 8 rupees each. Riya buys 6 pencils and 2 erasers "
            "at 5 rupees each. She pays with a 100 rupee note. How much change does she get?")
r["answer"] # 42
r["agreement"] # share of tries that gave this answer
r["confidence"] # checker-weighted share of support for this answer
r["program"] # the program behind the answer
```

- solve(q, tries=1) writes one greedy program and does not use the checker. It is the fastest setting.
- solve(q, tries=8), the default, samples 8 programs at temperature 0.7, runs them all and returns the answer with the most checker-weighted support.
- The result also lists every program, the value it returned and its checker score.
- From the shell: `lilavati "<question>" --tries 8 --show-program`.

Prompt formats. To use the model directly, the prompt is:

```
<bos>Question: {question}
Answer (Python):
```python
```

The model continues with a function and closes the code block. The checker's input is the same prompt followed by the program, a closing fence, Result: {value, or "error" if the program failed} and Correct:. P(correct) is the softmax over the logits of the single tokens “yes” and “no” at the last position. Inputs longer than 768 tokens keep their last 768 tokens.

### 3. Architecture

Lilavati is a decoder-only transformer. Its `config.json` declares the standard `Gemma4ForCausalLM` class of transformers (model type `gemma4_text`, version 5.18 or newer), so no custom modelling code is needed. The details below are those in the released `config.json`.

Lilavati and Lilavati-Checker	
Layers	12
Hidden width	640
Attention heads	10 query heads and 2 key-value heads (grouped-query attention)
Head size	64 in sliding-window layers, 128 in global layers
Local and global attention	Three sliding-window layers, then one global layer, repeated: layers 4, 8 and 12 attend over the whole input, the others over a window of the last 512 tokens
Position encoding	Rotary, with base 10,000 in sliding-window layers and 1,000,000 in global layers
MLP	Gated, GELU (tanh approximation), width 2,560
Output	Final-logit soft-cap of 15
Vocabulary	32,768 tokens
Embeddings	Tied: one table of $32,768 \times 640$ (21.0M parameters) serves as both the input embedding and the output projection
Parameters	94,732,812 in total (94.7M), of which 73.8M are outside the embedding table
Context	Trained at 4,096 tokens. The configuration allows 16,384 positions, but inputs longer than 4,096 tokens are untested

Tying the embeddings matters at this size: the embedding table is a large share of all parameters, and sharing it leaves more of the budget for the transformer layers. Word problems and their programs are far shorter than the trained context.

Lilavati-Checker has the same architecture and tokenizer. It has no separate classification head: it reuses the language-model output layer and reads its judgement from the probabilities of the next tokens “yes” and “no”. The checker therefore loads and runs with exactly the same code as the generator.

Weights are stored in bf16. The package runs them in bf16 on NVIDIA GPUs and in fp32 on CPUs and Apple-silicon GPUs.

### 4. Training (high level)

Both models were trained by Udgam Labs; neither starts from another organisation’s weights.

**Lilavati** was trained in two phases:

1. **Pretraining from random initialisation** on public synthetic maths datasets with Python solutions, together with general English text:
  - TinyGSM (MIT licence): about 12M grade-school problems with Python solutions;
  - `tinygsm_mixtral_12M`: TinyGSM-style problems with solutions written by Mixtral;
  - general English web and educational text.
2. **Further training on word problems with execution-verified Python solutions:** the GSM8K training split (MIT licence), and problems generated with large language models whose solutions were verified by running them.

**Lilavati-Checker** was trained to judge Lilavati’s programs. It saw only training-set problems, never test-set problems, and only programs in Lilavati’s style.

**Test-set hygiene.** No test-set problems were used to train either model. Every pretraining document was checked against every evaluation set for shared 13-word sequences (section 6.6). Problems added in the second phase were filtered against all the evaluation sets for shared 8-word sequences.

This report does not describe data proportions, training settings or compute.

## 5. Inference

### 5.1 Writing programs

Given the prompt in section 2, Lilavati writes a Python function, usually named `solution()` or `simple_math_problem()`, and ends it with a closing code fence, where generation stops. For one try the package decodes greedily. For several tries it samples at plain temperature 0.7, with top-k and top-p turned off. This is also the setting of every many-try result in section 6.

### 5.2 Running programs safely

Model-written code needs care, especially in a public demo where a visitor may try to steer the model into writing something harmful. The package's runner (`lilavati/safe_exec.py`) has three layers:

1. **A structure check before running.** The program is parsed and allowed only if it uses plain arithmetic, variables, functions, loops, conditions, lists and dictionaries, f-strings, a short list of safe built-in functions, and the `math` module. It is rejected if it imports anything else, uses names that start with a double underscore or attributes that start with an underscore, defines classes, uses `with` blocks, or calls functions such as `open`, `exec`, `eval` or `getattr`. Programs that define no function, and very long or very large programs, are also rejected.
2. **A separate, limited process.** An accepted program runs in a fresh Python interpreter with a minimal set of built-ins and with limits on CPU time, memory, file size, open files and child processes. On macOS it also runs under the system sandbox (`sandbox-exec`), with network access and file writes denied.
3. **A wall-clock timeout** of 3 seconds.

A try produces an answer only if its function returns a finite number. A rejected program, a crash, a timeout or a non-numeric return value counts as a failed try.

The runner is designed for maths programs from this model, not as a general-purpose sandbox. On systems other than macOS only the process limits apply. A public service should add its own isolation, such as a container or a virtual machine.

### 5.3 Combining tries

With  $N$  tries, each successful try returns a number. The package groups equal numbers (rounded to six decimal places) and can choose an answer in three ways:

- **Majority vote:** the number returned by the most tries.
- **Checker-weighted vote (the default):** each candidate number receives the sum of the checker's  $P(\text{correct})$  over all the programs that returned it, and the number with the largest sum wins.
- **Checker top pick:** the single program with the highest  $P(\text{correct})$  supplies the answer. With 48 tries this is TinyGSM's verifier protocol, which we call `verify48@1`.

Every many-try answer comes with two signals:

- **agreement:** the share of all tries that returned the winning number;
- **confidence:** the winning number's share of the total checker weight.

When either is below 0.5, the answer should be treated as a guess, and the public try box then says that it is not sure. Section 7.6 measures how well these signals separate right from wrong answers. If every try fails, `solve()` returns no answer rather than a guess. With one try there is no checker score, so confidence is empty.

### 5.4 Speed and memory

Every extra try costs one more generated program and one more checker pass, so compute grows roughly in proportion to the number of tries. The default of 8 tries balances speed and accuracy; section 6.3 gives results for 8, 16 and 48 tries.

We timed the public `solve()` on an Apple M6 Mac mini (12 CPU cores, 24 GB of memory; macOS 27.0.1, Python 3.12, torch 2.14.1, transformers 5.20.0) on 17 October 2026. Each setting ran in a fresh process

with the package defaults: fp32 weights, both models loaded, and on the Apple GPU (MPS) the checker kept on the CPU, as the package does on Macs. PyTorch used its default of 2 CPU threads. After two untimed warm-up questions, each of the first 20 GSM8K test problems was solved one at a time, and each call was timed end to end: writing the programs, running them in the sandbox, the checker and the vote.

Setting	Device	Median seconds per question	90th percentile	Peak memory (RSS)
<code>solve(q, tries=1)</code>	CPU	1.85	2.93	0.89 GB
<code>solve(q, tries=8)</code>	CPU	5.19	6.79	1.31 GB
<code>solve(q, tries=1)</code>	Apple GPU (MPS)	0.86	1.07	0.90 GB
<code>solve(q, tries=8)</code>	Apple GPU (MPS)	1.75	2.50	1.08 GB

Peak memory is the Python process’s maximum resident set size; the short-lived sandbox processes are not included, and on MPS the Metal driver also reported about 1.0 GB of GPU allocations in the same unified memory. The computer was running other jobs at the time (load average 6 to 8), so these timings are conservative. Loading both models took under 3 seconds once they were downloaded. On the free CPU tier of the public demo Space, a default 8-try answer takes about 17 seconds.

## 6. Evaluation

### 6.1 Setup

All numbers in this section were measured by Udgam Labs unless they are labelled as reported by others.

- **Scoring.** The model’s program is run and its return value is compared with the reference answer. Whole-number answers must match exactly. Decimal answers must match to the precision given in the reference.
- **Test sets:**
  - GSM8K test (Cobbe et al., 2021): 1,319 problems.
  - GSM-Plus testmini (Li et al., 2024): the 7 perturbation types that have a numeric answer, 2,100 problems (300 per type). The eighth type, “critical thinking”, has no numeric answer and is left out.
  - SVAMP (Patel et al., 2021): all 1,000 problems, train and test splits together.
  - ASDiv (Miao et al., 2020): the 2,159 problems with numeric answers.
  - GSM-Hard (Gao et al., 2022): the 1,319 GSM8K test problems with larger numbers substituted.
- **Decoding.** One try is greedy. Many tries are sampled at temperature 0.7, with no top-k or top-p cut-off.
- **Sampling uncertainty.** On 1,319 problems, a score near 55% has a 95% binomial interval of about  $\pm 2.7$  points. On a 300-problem subset the interval is about  $\pm 5.6$  points. Smaller differences should not be over-read.

**Two harnesses.** The published numbers come from our training-time harness, which runs the model in bf16 on an NVIDIA GPU. We also scored the released weights end to end with the public code:

- Before release, `eval/gsm8k.py` on a CPU gave 726 of 1,319 GSM8K test problems, or 55.0%, with one greedy try (57.7% on the first 300 problems, against 57.3% in the training-time harness).
- After release, on 17 October 2026, we re-ran every one-try set with the public package on the Mac described in section 5.4 (fp32 on the Apple GPU), using the released Hugging Face files (their SHA-256 checksums match the repository’s SHA256SUMS). Problems of similar length were batched for speed; on 120 test problems the batched programs were identical to those from one-at-a-time `solve(q, tries=1)`, and the GSM8K total was again 726 of 1,319.

The two harnesses agree on whether the answer is right for 1,305 of the 1,319 GSM8K problems. The small differences come from numerical precision (bf16 against fp32). Both sets of numbers are given below.

## 6.2 One try

Test set (one greedy try, code executed)	Items	Training-time harness (bf16, GPU)	Public code (fp32, Apple GPU)
GSM8K test	1,319	54.7	55.0
GSM-Plus testmini, 7 answerable types	2,100	39.2	39.3
... of which “distraction insertion” (an irrelevant fact is added)	300	20.7	22.3
SVAMP	1,000	69.9	70.2
ASDiv, numeric answers	2,159	74.0	73.7
GSM-Hard	1,319	43.7	43.4

Section 7.2 gives all seven GSM-Plus perturbation types.

## 6.3 Many tries with the checker

GSM8K test. Lilavati wrote 48 programs per problem at temperature 0.7 in the training-time harness, and each program was run. The 8-try and 16-try columns use the first 8 and the first 16 of each problem’s 48 programs.

How the answer is chosen	8 tries	16 tries	48 tries
<b>Checker-weighted vote</b> (the <code>solve()</code> default)	65.4	68.8	<b>72.3</b>
The checker’s single top-rated program ( <code>verify@1</code> )	64.1	65.9	66.7
Plain majority vote (no checker)	61.2	63.1	65.7
At least one program is right ( <code>pass@k</code> , an upper bound)	75.3	82.3	89.5

For reference, one greedy try scores 54.7.

**Which programs count.** The first 8 and first 16 programs happen to score at the low end. Split into disjoint groups, the same 48 programs give a mean checker-weighted vote of 67.5 at 8 tries (six groups, range 65.6 to 68.4) and 69.9 at 16 tries (three groups, range 68.8 to 70.4). The table keeps the first-k figures, which are the ones published.

**Check with the released checker.** The training-time harness saved all 63,312 programs with their results. We re-scored them on 17 October 2026 with the released Lilavati-Checker, using the package’s checker input, in fp32 on the Apple GPU of the Mac in section 5.4. The checker-weighted vote was identical at 16 and 48 tries (68.8 and 72.3) and 65.6 at 8 tries (training-time 65.4); the top pick was identical at 8 and 16 tries (64.1 and 65.9) and 66.8 at 48 tries (training-time 66.7). These differences come from numerical precision. The majority-vote and `pass@k` rows do not use the checker and reproduce exactly from the saved programs.

**The package default on its own.** We also ran `solve(q)` with its defaults (8 fresh tries at temperature 0.7 and the checker-weighted vote) on all 1,319 GSM8K test problems with the public package on the same Mac: generator on the Apple GPU, checker on the CPU, random seed 1234, with problems of similar length sampled in batches. It scored 68.4%. On the same programs a plain majority vote scores 61.6%, and for 76.7% of problems at least one of the 8 programs is right. This is within the range of the saved programs’ disjoint groups of 8.

## 6.4 Comparison with other open models up to 150M parameters

We measured these models on 14 and 15 October 2026 with one evaluation harness, zero-shot and greedy, each with its own chat template and the instruction to reason step by step and put the final answer in `\boxed{}`. The other models answer in text, their native format, and are scored on the boxed answer, or on the last number if there is none. Lilavati's scores are with code executed (section 6.2). Five of the models were scored on the first 300 GSM8K test items only; Lilavati's score on the same 300 items is given for comparison. The output cap differed between models and is given in the notes.

Model	Params	GSM8K test	Notes
Lilavati	95M	54.7 (57.3 on the first 300)	code executed
cagliostro-v3.5-chat	146M	21.3	first 300 items; 1,024-token output cap
Falcon-H1-Tiny-R-90M	90M	18.7	all items; 16,384-token output cap, which 53% of its answers still reached. SVAMP: 40.1 with an 8,192-token cap (Lilavati 69.9)
MobileLLM-R1.5-140M	140M	6.7	first 300 items; 99% of its answers ran past the 8,192-token cap. Its developers report 8.3
GPT-X3-150M	150M	2.3	first 300 items; 1,024-token cap, reached by 42% of its answers
GPT-X2.5-135M	135M	1.7	first 300 items; 1,024-token cap, reached by 99% of its answers
Supra2-100M-Instruct	100M	1.6	all items; 2,048-token context, no answers cut off
JugnuLM-110M-R2plus	110M	1.3	first 300 items; 1,024-token cap, reached by 90% of its answers

The strongest of these, cagliostro-v3.5-chat, scores 21.3% on the first 300 items. On the same items Lilavati scores 57.3%, about  $2.7\times$  ( $57.3 / 21.3$ ). The  $2.7\times$  is computed on these 300 items only. Each of these two scores has a sampling uncertainty of several points (section 6.1), so the ratio is approximate.

Three caveats apply:

- **This compares systems, not only models.** Lilavati's arithmetic is done by Python; the other models must compute in text.
- **Output caps.** Where many of a model's answers reached its cap, its score is a lower bound for that model under longer generation.
- **It covers the models we could find and run.** Three further recent models in this range (nanowhale-100m, Surjo-100m and lille-130m) could not be run in our harness at the time: one ran out of GPU memory, and our inference libraries could not load the architectures of the other two. lille-130m-instruct reports 7.73 on GSM8K on its own model card (self-reported; settings not stated). A scan on release day found one newer model in the range with a reported GSM8K score: Inferno-Exp-1.5-GSM8K (76M) reports 8.87% on its model card (zero-shot, greedy, all 1,319 items). We did not measure it.

## 6.5 Comparison with TinyGSM's published results

TinyGSM (Liu et al., 2023) is the closest published work. Its models were not released, so the figures below are reported by its authors, not measured by us, and they come from a different evaluation pipeline.

System	Size	One try	verify48@1
TinyGSM generator, pretrained on Phi-1.5 data before training on TinyGSM (reported)	125M, plus a 125M verifier	63.1	68.9
TinyGSM generator trained from random initialisation (reported)	125M	53.1	–
Lilavati with Lilavati-Checker (measured)	95M, plus a 95M checker	54.7	66.7

Under TinyGSM's exact protocol, the verifier's single top pick out of 48 programs, Lilavati with its checker scores 66.7%. That is below TinyGSM's reported 68.9%. With one greedy try, Lilavati (54.7%) is also below TinyGSM's pretrained 125M generator (63.1%) and close to its 125M generator trained from random initialisation (53.1%). The checker-weighted vote (72.3%) uses a different way of choosing the answer, so we do not set it against TinyGSM's verify48@1 figure.

## 6.6 Contamination audit

Every document Lilavati was pretrained on was checked against each test set for shared 13-word sequences:

Test set	Items sharing a 13-gram with the pretraining data
GSM8K test	0 of 1,319
GSM-Hard	0 of 1,319
GSM-Plus testmini	20 of 2,400 (0.8%); 19 of the 2,100 scored items
SVAMP	22 of 1,000 (2.2%)
ASDiv	154 of 2,159 (7.1%)

To see whether the overlap inflates the scores, we compare each set’s score on the items with no overlap (the clean subset) with its score on the full set. The released model was scored in both harnesses; the pretrained checkpoint, before the second training phase, is shown for comparison.

Test set (one greedy try)	Released model, full set	Released model, clean subset	Pretrained checkpoint, full set	Pretrained checkpoint, clean subset
GSM-Plus, 7 types (2,100; clean 2,081)	39.2 (39.3)	39.1 (39.2)	37.7	37.6
SVAMP (1,000; clean 978)	69.9 (70.2)	69.6 (70.0)	68.8	68.7
ASDiv (2,159; clean 2,005)	74.0 (73.7)	75.3 (75.0)	73.7	75.1

The first number in each released-model cell is from the training-time harness and the number in brackets from the public code. Removing the overlapping items lowers the GSM-Plus and SVAMP scores by at most 0.3 points and raises the ASDiv score by 1.2 to 1.4 points, for both checkpoints. The overlapping items themselves are few: on GSM-Plus and SVAMP the released model solves a larger share of them than of the rest (10 of 19 and 18 of 22 in the training-time harness), and on ASDiv a smaller share (88 of 154). Because removing them barely moves any full-set score, this kind of overlap does not explain the results. A 13-gram check finds near-copies, not reworded problems, so it cannot rule out problems that were rewritten in other words. GSM8K and GSM-Hard have no 13-gram overlap at all. Problems added in the second phase were filtered against all of these test sets for shared 8-word sequences.

## 7. Analysis

### 7.1 What the extra tries buy

- **Voting alone:** from one greedy try (54.7) to a plain majority vote over 48 tries (65.7) is +11.0 points. This is consistent with wrong programs tending to return different wrong numbers while right programs agree.
- **The checker on top of voting:** from the majority vote to the checker-weighted vote is +4.2 points at 8 tries (61.2 to 65.4), +5.7 at 16 (63.1 to 68.8) and +6.6 at 48 (65.7 to 72.3). The checker adds information beyond agreement, and more of it as the number of tries grows.
- **Weighting does better than picking.** The checker’s single top pick (66.7) is 5.6 points below the weighted vote at 48 tries. Going from 16 to 48 tries lifts the weighted vote by 3.5 points (68.8 to 72.3) and the majority vote by 2.6 (63.1 to 65.7), but the top pick by only 0.8 (65.9 to 66.7). At this size the checker’s rating of any one program is noisy, and pooling ratings over all the programs that return the same number is more reliable.
- **Room left:** for 89.5% of problems at least one of the 48 programs is right (82.3% at 16 tries). The 17.2-point gap between this upper bound and the weighted vote at 48 tries (13.5 points at 16 tries) is the room left for better answer selection.

## 7.2 Robustness

- **GSM-Hard: 43.7, against 54.7 on GSM8K (–11.0).** GSM-Hard keeps GSM8K’s problems but substitutes larger numbers. Python does the arithmetic, so larger numbers should cost little in computation. Our reading is that unfamiliar number sizes also change how the model reads or plans a problem. We have not audited GSM-Hard’s labels, so part of the gap may come from the set itself.
- **GSM-Plus: 39.2 over the 7 answerable types (–15.5 against GSM8K).** The seven types, 300 problems each, one greedy try:

GSM-Plus perturbation type	Training-time harness	Public code
Numerical substitution	52.7	52.3
Digit expansion	52.0	52.0
Problem understanding (the problem is reworded)	52.0	51.7
Integer-decimal-fraction conversion	42.0	41.7
Reversing operation (asks for a quantity that was given)	27.7	27.7
Adding operation (one more step is needed)	27.7	27.3
Distraction insertion (an irrelevant fact is added)	20.7	22.3
<b>All 7 types</b>	<b>39.2</b>	<b>39.3</b>

Changing the numbers or rewording the problem costs about 2 to 3 points against GSM8K’s 54.7. Fractions and decimals cost about 13. Problems that must be worked backwards, need one more step, or contain an irrelevant fact cost 27 to 34 points.

- **SVAMP (69.9) and ASDiv (74.0)** score higher than GSM8K.

## 7.3 Failure types

The model card lists these weaknesses:

- **Irrelevant information:** a number that should be ignored often gets used anyway.
- **Working backwards,** and problems that **add an extra step** to a familiar pattern.
- **Unit conversions** and some **ratio** questions.
- **Unanswerable questions:** it always returns a number, even when the question has no answer. For this reason GSM-Plus’s “critical thinking” type can never be scored as correct and is left out of section 6.
- **Anything that is not a maths word problem.** It is not a chatbot and has little general knowledge.

Three of these correspond directly to GSM-Plus perturbation types, and they are its three lowest-scoring types (section 7.2): distraction insertion, reversing an operation, and adding an operation.

The public release page shows three real failures with the default 8 tries:

Type	Question (shortened)	Lilavati’s answer	Right answer
Unit conversion	How many seconds are there in 3.5 hours?	756,000	12,600
Ratio	3 cups of rice serve 4 people; how many cups for 10 people?	30	7.5
Working backwards	After spending half his money on a book and then 30 rupees on lunch, Arjun has 70 rupees left. How much did he have at first?	340	200

The first wrong answer is consistent with one conversion step too many ( $3.5 \times 60 \times 60 \times 60$ ), and the second with ignoring “for 4 people” ( $3 \times 10$ ). In all three cases only 1 of the 8 tries agreed with the answer given, so the try box said that it was not sure. Low agreement is the signal to look for (section 7.6).

## 7.4 When the checker is wrong

The checker can be confidently wrong, especially when most tries share the same mistake, such as using a number that should have been ignored. Both voting and the checker reward agreement, so a systematic error can survive the selection step. The checker was trained only on programs in Lilavati's style, and its ratings of programs written by other models are untested.

## 7.5 Problem length and step count

Code execution removes computation errors but not planning errors, so we expected the remaining errors to concentrate in problems that need longer plans. To test this we counted the steps in each GSM8K reference solution: the number of `<<...>>` calculation annotations, or, for the 18 solutions without annotations, the number of non-final lines. The scores are from the public-code greedy run (55.0% overall).

Steps in the reference solution	Problems	Lilavati, one greedy try
1–2	427	68.9
3	367	59.4
4	295	48.5
5 or more	230	30.9

  

Question length (words; quartiles of the test set)	Problems	Lilavati, one greedy try
Up to 33	350	73.1
34–43	330	63.3
44–55	316	48.1
56 or more	323	33.7

Accuracy falls steadily with both the number of steps and the length of the question: problems that need five or more steps are solved less than half as often as those that need one or two. Step count and length are correlated, so the two tables describe overlapping effects. This supports the reading that the remaining errors are mostly errors of planning.

## 7.6 Agreement and confidence as warning signs

With several tries, `solve()` returns the answer's agreement and confidence (section 5.3). We measured how well they separate right from wrong answers in the package-default run of section 6.3 (8 tries, all 1,319 GSM8K test problems, 68.4% right overall):

Signal (8 tries)	Share of questions	Accuracy
Agreement $\geq 0.5$	59.7%	90.0
Agreement $< 0.5$	40.3%	36.3
Confidence $\geq 0.5$	90.8%	72.5
Confidence $< 0.5$	9.2%	27.3
Both $\geq 0.5$	59.6%	90.1
Either $< 0.5$ (the try box says it is not sure)	40.4%	36.4

The saved training-time programs (the first 8 of 48 per problem, scored with the released checker) show the same pattern: agreement  $\geq 0.5$  covers 58.7% of questions at 90.3% accuracy and agreement  $< 0.5$  covers 41.3% at 30.5%; confidence  $\geq 0.5$  covers 90.5% at 70.8% and confidence  $< 0.5$  covers 9.5% at 16.0%.

Agreement is the sharper signal. When all 8 tries agree (23.4% of questions) the answer is right 96.8% of the time; when fewer than a quarter agree (20.7%), 22.3%. Confidence below 0.5 is rare, because the checker-weighted winner usually holds most of the weight, and a confidence just above 0.5 is weak evidence: answers with confidence between 0.5 and 0.75 are right only 41.7% of the time. The 0.5 thresholds therefore flag about 40% of answers, of which about a third are right. They are a useful warning, not a guarantee in either direction.

## 8. Limitations

- **Scope.** Lilavati is trained for English grade-school maths word problems. The training problems are mostly in that style, so expect lower accuracy on other styles, longer problems and other languages. It is not a chatbot, a proof assistant or a solver for competition maths.
- **It always answers.** With one try there is no abstention and no checker score. With several tries, treat answers whose agreement or confidence is below 0.5 as guesses: with the default 8 tries on GSM8K test, 40% of answers fell below one of these thresholds and only 36% of those were right, against 90% for the rest (section 7.6).
- **A program that runs is not proof that it is correct.** Lilavati is not suitable for grading, financial or medical calculations, or anything where a wrong number could cause harm, without a person checking the program.
- **Known weak spots** are listed in section 7.3, and accuracy falls with the number of steps a problem needs (section 7.5).
- **Checker.** It judges grade-school word problems only. It is not a general code or proof verifier, and a high rating is not a guarantee.
- **Sandbox.** The system sandbox layer exists only on macOS; a public service needs its own isolation (section 5.2).
- **Evaluation.**
  - Lilavati’s scores are with code execution, while the compared small models answer in text.
  - The comparison models were all run in one harness, but five of them were scored on a 300-item subset, and their output caps differ (section 6.4).
  - TinyGSM’s figures are reported by its authors and not reproduced by us.
  - Results come from one training run; we have not measured run-to-run variance.
  - The robustness sets were measured with one greedy try only.
  - The many-try figures depend on which sampled programs are used; disjoint groups of 8 programs vary by about 3 points (section 6.3).
  - The contamination audit looks for shared 13-word sequences, so it finds near-copies but not reworded problems.
  - The speed figures come from one computer that was running other jobs at the time.
- **Context.** The model was trained at 4,096 tokens. The configuration allows 16,384 positions, but inputs longer than 4,096 tokens are untested.
- **Machine-generated data.** Much of the training data was written by other language models and can carry their errors and stylistic habits.

## 9. Release details

Item	Where
Lilavati: weights and model card	<a href="https://huggingface.co/UdgamLabs/Lilavati">huggingface.co/UdgamLabs/Lilavati</a>
Lilavati-Checker: weights and model card	<a href="https://huggingface.co/UdgamLabs/Lilavati-Checker">huggingface.co/UdgamLabs/Lilavati-Checker</a>
Code, evaluation scripts and checksums (v1.0.0)	<a href="https://github.com/UdgamLabs/lilavati">github.com/UdgamLabs/lilavati</a>
Try box (no login)	<a href="https://udgamlabs.com/models/lilavati">udgamlabs.com/models/lilavati</a>
Demo Space (free CPU)	<a href="https://huggingface.co/spaces/jiteshkprasad/Lilavati">huggingface.co/spaces/jiteshkprasad/Lilavati</a>

- **Release date:** 16 October 2026; version 1.0.0. The models are also grouped in a public “Lilavati” collection on the Udgam Labs Hugging Face page.
- **Requirements:** Python 3.10+, torch 2.3+ and transformers 5.18+. No login is needed to download the weights.
- **Files:** bf16 safetensors, about 190 MB per model; SHA256SUMS in the GitHub repository.
- **Licence:** Apache-2.0 for the weights and the code. The public training datasets carry their own licences (TinyGSM and GSM8K: MIT).
- **Developer:** Udgam Labs, [udgamlabs.com](https://udgamlabs.com).
- **Contact:** the Community tab of the Hugging Face model pages, or issues on the GitHub repository.

```

title = {Lilavati: a 95M pocket mathematician that solves word problems by writing Python},
author = {Udgam Labs},
year = {2026},
url = {https://huggingface.co/UdgamLabs/Lilavati}
}

```

## 10. Conclusion

A 94.7M-parameter model trained from scratch solves more than half of GSM8K’s test problems with one try when it writes Python and the program is executed. A same-size checker, used for weighted voting over 48 tries, lifts this to 72.3%. Under TinyGSM’s verifier protocol Lilavati scores 66.7%, below the 68.9% that TinyGSM reports for its unreleased 125M models. On the first 300 GSM8K test items, its greedy score is about  $2.7\times$  that of the best other open model of up to 150M parameters we measured, which answers in text.

The clearest limits are irrelevant facts, working backwards, extra steps, unit conversions and questions with no answer, and accuracy falls steadily as problems need more steps. Since 89.5% of problems have at least one right program among 48 tries, better answer selection is the largest remaining lever. The weights, the safe runner and the evaluation scripts are open under Apache-2.0, so the results can be checked and the model can serve as a starting point for fine-tuning.

## 11. References

1. Liu, B., Bubeck, S., Eldan, R., Kulkarni, J., Li, Y., Nguyen, A., Ward, R. and Zhang, Y. (2023). *TinyGSM: achieving >80% on GSM8k with small language models*. [arXiv:2312.09241](#).
2. Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C. and Schulman, J. (2021). *Training Verifiers to Solve Math Word Problems*. [arXiv:2110.14168](#). (GSM8K.)
3. Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J. and Neubig, G. (2022). *PAL: Program-aided Language Models*. [arXiv:2211.10435](#). (Program-aided reasoning; GSM-Hard.)
4. Chen, W., Ma, X., Wang, X. and Cohen, W. W. (2022). *Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks*. [arXiv:2211.12588](#).
5. Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A. and Zhou, D. (2022). *Self-Consistency Improves Chain of Thought Reasoning in Language Models*. [arXiv:2203.11171](#).
6. Li, Q., Cui, L., Zhao, X., Kong, L. and Bi, W. (2024). *GSM-Plus: A Comprehensive Benchmark for Evaluating the Robustness of LLMs as Mathematical Problem Solvers*. [arXiv:2402.19255](#).
7. Patel, A., Bhattamishra, S. and Goyal, N. (2021). *Are NLP Models really able to Solve Simple Math Word Problems?* [arXiv:2103.07191](#). (SVAMP)
8. Miao, S.-Y., Liang, C.-C. and Su, K.-Y. (2020). *A Diverse Corpus for Evaluating and Developing English Math Word Problem Solvers*. ACL 2020; [arXiv:2106.15772](#). (ASDiv.)
9. Datasets on Hugging Face: [TinyGSM/TinyGSM](#), [reciprocate/tinygsm\\_mixtral\\_12M](#), [openai/gsm8k](#), [qintongli/GSM-Plus](#), [Chilled/SVAMP](#), [EleutherAI/asdiv](#), [reasoning-machines/gsm-hard](#).
10. Compared models on Hugging Face:
  - measured: [bench-labs/cagliostro-v3.5-chat](#), [tiiuae/Falcon-H1-Tiny-R-90M](#), [facebook/MobileLLM-R1.5-140M](#), [AxiomicLabs/GPT-X3-150M](#), [AxiomicLabs/GPT-X2.5-135M](#), [SupraLabs/Supra2-100M-Instruct](#), [altslate/JugnuLM-110M-R2plus](#);
  - not run: [HuggingFaceTB/nanowhale-100m](#), [SurjoLabs/Surjo-100m](#), [Nikity/lille-130m-instruct](#), [AlgoDriveAI/Inferno-Exp-1.5-GSM8K](#).
11. Bhāskara II (1150 CE). *Līlāvati*.