SYNTHETIC ERROR INJECTION FAILS TO ELICIT SELF-CORRECTION IN LANGUAGE MODELS

David X. Wu*, **Shreyas Kapur***, **Anant Sahai & Stuart Russell**
Department of EECS, UC Berkeley
Berkeley, CA, USA
{david_wu,srkp}@berkeley.edu

ABSTRACT

Reasoning language models trained with reinforcement learning have been observed to exhibit emergent self-correction behavior. Inspired by the success of synthetic error injection in autonomous driving and robotics, we attempt to directly teach models to self-correct without reinforcement learning. Our approach inserts artificial errors into reasoning chains, masks them, and supervises the model to recognize and correct these mistakes. Despite the intuitive appeal of this method, we find that it fails to significantly improve performance even on simple synthetic tasks across multiple models. Moreover, even when the model catches its own error, it often parrots the original mistake. We find that the distribution shift of synthetic errors to on-policy errors significantly degrades the error-correction capabilities of the fine-tuned model, even with good synthetic coverage of on-policy errors. Our results help explain why on-policy reinforcement learning methods have proven uniquely effective for eliciting self-correction.

1 INTRODUCTION

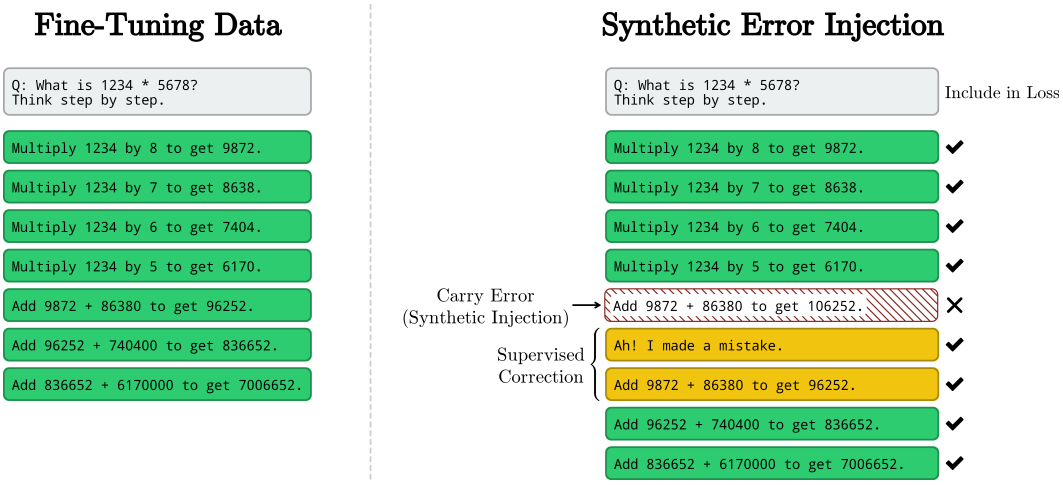


Figure 1: An illustration of the synthetic error injection process. The (Left) panel shows a correct Chain-of-Thought (CoT) trace used for standard fine-tuning (FT). The (Right) panel details the Error Injection Fine-Tuning (EIFT) methodology, where a correct step from a golden CoT is replaced by a three-part sequence: (1) a synthetically injected erroneous step (e.g., a "Carry Error"), (2) an explicit error recognition step ("Ah! I made a mistake."), and (3) the original correct step as a supervised correction. As indicated by the "Include in Loss?" column, the loss is computed for the recognition and correction steps but masked for the synthetically injected error.

The current paradigm for eliciting reasoning in large language models is reinforcement learning (RL) (Guo et al., 2025). These methods have driven substantial gains in mathematics and coding benchmarks across both closed and open-source models (Jaech et al., 2024; Comanici et al., 2025;

*Equal contribution.

Bai et al., 2025; Lambert et al., 2024). Such systems often generate an extended chain of thought (CoT) before producing a final answer; within these CoTs, we observe emergent self-reflection and error correction, as the model outlines a solution strategy and then identifies and revises mistakes.

Is it possible to obtain these self-reflection abilities without resorting to reinforcement learning, which is often computationally expensive?

In this paper, we adopt a similar idea for reasoning language models (Figure 1). We use a purely supervised approach: insert synthetic errors into the CoT, mask them, and add a supervised correction step so the model learns to recognize and recover from its own mistakes.

Surprisingly, even on toy tasks, the method fails to yield significant performance improvements across all tested models. We explore two potential hypotheses:

(i) We investigate the "solver-verifier gap"—the difference in difficulty of generating and checking a step—but find that the method's failure persists even when checking is significantly easier.

(ii) We propose that the true issue is a distribution mismatch. While our synthetic error distribution successfully covers the overall distribution of real errors, the model only learns to reliably correct synthetic errors. It fails to generalize this error-correction capability to the distribution of context-dependent errors that the base model itself is prone to making.

Our contributions in this paper are therefore twofold. First, we formally test the hypothesis that supervised error injection can elicit self-correction in LLMs, and we demonstrate its empirical failure. Second, our analysis pinpoints the reason for this failure: a critical mismatch between synthetic error distributions and the model's own innate error modes. This finding suggests that successful self-correction requires more precisely matching error distributions: it must target the specific, context-dependent failures a model is prone to, suggesting why on-policy methods like RL have proven uniquely effective.

2 Experimental setup

We trained our models on a cluster of A100 and A6000 GPUs. For each model in {`Qwen2.5-Math-1.5B-Instruct`, `gemma-3-1B-it`, `Llama-3.2-1B-Instruct`} and task in {`mult`, `sudoku`}, we trained two versions of the model which differed only in whether the data contained error-injection CoTs. To solidify the terminology, we define the model variants as follows.

Definition 2.1 (FT model)**.** *We refer to the model trained only on clean CoTs as the* FT model *(Fine Tuned).*

Definition 2.2 (EIFT model)**.** *We refer to the model trained on error injected CoTs as the* EIFT model *(Error Injection Fine Tuned).*

The details of the EIFT training methodology and data mix can be found in Section A.1; see Section B for training hyperparameters.

3 Results

For evaluation, we measured accuracy and error correction/recognition rates using a fixed set of 1000 randomly sampled problems. In all the figures below, we also report $1.96 \cdot \text{SE}$ error bars.

We hypothesized that the EIFT model would have better performance than the corresponding FT model. In Figure 2 we see that this hypothesis is not strongly supported by the empirical results. For multiplication, we see modest to no performance gains from EIFT training. For Sudoku, we see more nontrivial gains of roughly 10% for `Qwen2.5` and `Llama-3.2`. However, `gemma-3` shows no performance boost from EIFT training. The baseline performance of the FT model varied drastically depending on the base model and task. For example, `Qwen2.5` performed quite well (FT performance: 92%) on multiplication, but was not as strong on Sudoku. Conversely, `Llama-3.2` had very poor performance on multiplication (FT performance: 2.0%), likely due to how `Llama-3.2` tokenizes numbers, but had the best performance on Sudoku (FT performance:

52.2%). Apart from the issue of tokenization, we hypothesize that these fluctuations are due to the specific mid/post-training recipes for these models, e.g., since `Qwen2.5` is a specialized math model.

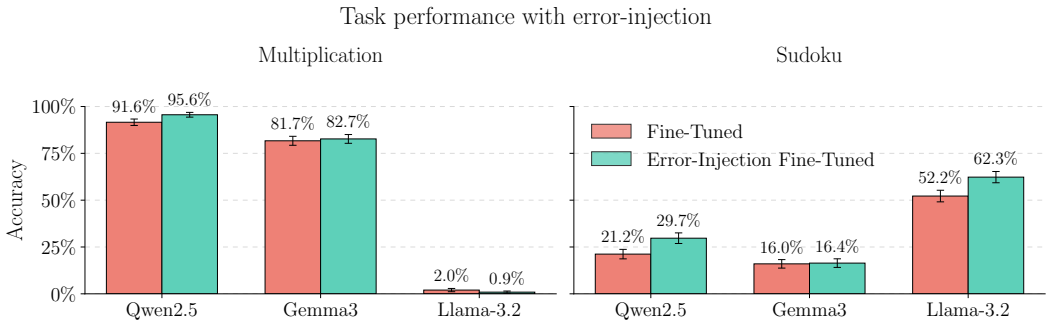


Figure 2: A comparison of model performance grouped by base model and task. We report the mean accuracy measured on a fixed set of 1000 randomly generated problems, and error bars are 1.96SE. The two model types are (1) FT (Fine-Tuned, red) models which are trained solely on golden CoTs or (2) EIFT (Error-Injection Fine-Tuned, green) on error-injected CoTs.

One plausible explanation for why the performance is not significantly boosted from EIFT training is that the model does not actually reliably correct its own errors. Thus, we turn to investigating the correction and recognition capabilities of the EIFT model.

In Figure 3 we see the EIFT model, despite having high error recognition rates for the synthetic errors (Definition A.2), suffers from a precipitous drop in correction and recognition rates when prompted with on-policy errors generated by the FT model (Definition A.3). In all cases except for `Qwen2.5` on multiplication, the FT error recognition rate drastically plummets—e.g., 94% $\rightarrow$ 8% for `Qwen2.5` on Sudoku, 83% $\rightarrow$ 20% for `gemma-3` on multiplication. Furthermore, there is a nontrivial gap in correction and recognition rates for FT errors. For Sudoku, this gap also exists for synthetic errors, suggesting that error correction is somewhat nontrivial for this task. For Qwen on multiplication, even though the model has better retention in error recognition (100% $\rightarrow$ 78%), it nevertheless significantly drops in error correction (99% $\rightarrow$ 40%).

Remark 3.1. *To be thorough, we also evaluated the FT models, and as expected the FT models did not possess error correction capabilities.*

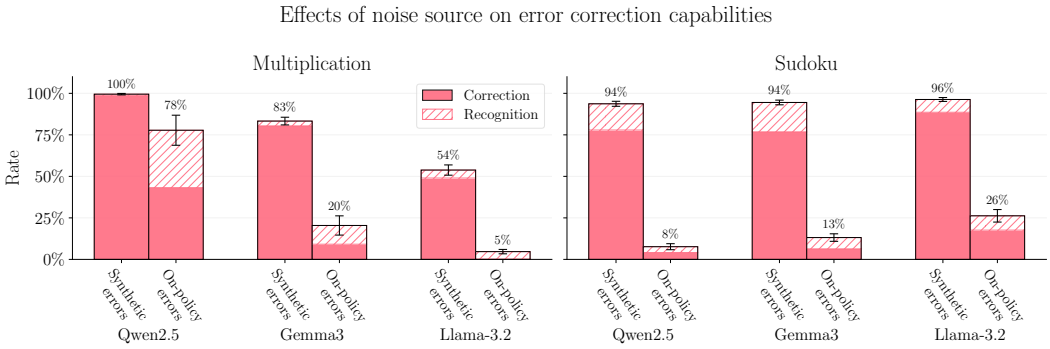


Figure 3: Comparison of error correction capabilities for EIFT models based on the error source and task. We report the mean correction and recognition rates measured on a fixed set of 1000 randomly generated problems, and error bars are 1.96SE. We compare errors that the EIFT model was trained to correct and recognize (Synthetic errors) versus errors the FT model naturally produces (On-policy errors). The recognition rates (hatched) are always larger than the correction rates (solid) as recognition is a superset of correction.

4 Discussion

We have shown that, even in toy synthetic settings, synthetic error-injected SFT does not recover the performance of online RL, in spite of having access to golden solutions and perfect verifiers. Our

results suggest that SFT-based approaches to error correction must essentially match the on-policy distribution, or else inject significantly richer information to condition the model to try different strategies.

REFERENCES

Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Mitchell Goff, Greg Hogan, George Hotz, Armand du Parc Locmaria, Kacper Raczy, Harald Schäfer, Adeeb Shihadeh, Weixing Zhang, and Yassine Yousfi. Learning to drive from a world model. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 1964–1973, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Hangfeng He, Hongming Zhang, and Dan Roth. Rethinking with retrieval: Faithful large language model inference. *arXiv preprint arXiv:2301.00303*, 2022.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, 2025.

OpenAI. OpenAI o3 and o4-mini system card. 2025.

Harald Schafer, Eder Santana, Andrew Haden, and Riccardo Biasini. A commute in data: The comma2k19 dataset, 2018.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 2550–2575, 2023.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie. Self-evaluation guided beam search for reasoning. *Advances in Neural Information Processing Systems*, 36:41618–41650, 2023.

Kaiyu Yang, Jia Deng, and Danqi Chen. Generating natural language proofs with verifier-guided search. *arXiv preprint arXiv:2205.12443*, 2022.

Xiao-Wen Yang, Xuan-Yi Zhu, Wen-Da Wei, Ding-Chu Zhang, Jie-Jing Shao, Zhi Zhou, Lan-Zhe Guo, and Yu-Feng Li. Step back to leap forward: Self-backtracking for boosting reasoning of language models. *arXiv preprint arXiv:2502.04404*, 2025.

APPENDIX

A RELATED WORK

Chain-of-thought (CoT) reasoning (Wei et al., 2022) is a popular approach for imparting reasoning abilities to language models. Recently, reinforcement learning (RL) (Jaech et al., 2024; OpenAI, 2025)—especially stabilization techniques such as GRPO (Guo et al., 2025)—have emerged as the dominant paradigm for teaching models to reason and to self-correct. There is growing interest in this direction, including work such as Kumar et al. (2024), which uses RL to train models to recognize and correct their own errors.

There has also been substantial effort to improve reasoning and self-correction abilities *without* RL (Shinn et al., 2023), given its computational cost. Approaches include using one LLM as a feedback model for another (Madaan et al., 2023; Schick et al., 2023), generate-then-rank strategies (He et al., 2022; Weng et al., 2023), and feedback engines that guide autoregressive decoding (Yang et al., 2022; Xie et al., 2023). Muennighoff et al. (2025), for instance, proposes a simple technique: append the token "Wait" multiple times when the model attempts to stop, thereby lengthening the reasoning chain and improving performance—without any reinforcement learning.

Kumar et al. (2024) also shows that offline–online distribution mismatch can yield poor on-policy self-correction. We examine this failure mode on even simpler tasks than `MATH` (Hendrycks et al., 2021), specifically small multiplication and Sudoku problems. By constructing a covering set of plausible modeling errors, we show that even such coverage is insufficient, strengthening their result.

Synthetic error injection is widely used in robotics (Goff et al., 2025; Schafer et al., 2018). The work most closely related to ours is Yang et al. (2025), which trains a language model to play Countdown and to backtrack in a tree-search harness using only supervised data and synthetic error injection. However, their method primarily distills the DFS procedure used to solve Countdown, rather than teaching the model to correct its own errors, and requires a costly sampling harness to generate solutions. In contrast, we teach the model to backtrack fully in a single linear CoT.

A.1 ERROR INJECTION

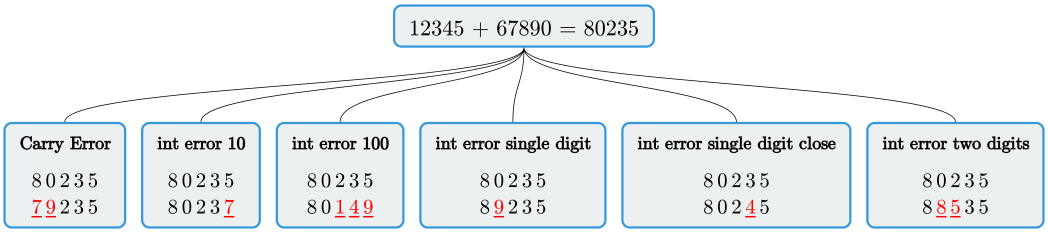


Figure 4: Types of errors we inject for the multiplication task. We take a correct step, depicted here as the addition step at the top of the figure. We then apply one of six types of errors with various probabilities; see Table 1 for more details. The red underlined digits in each box depict the result of applying the corresponding error type.

For the EIFT model, we used a data mix where 20% of the CoTs were completely correct, and 80% had 1 to 4 synthetic errors injected, where the number and locations of errors were sampled uniformly at random (and without replacement for locations). We found that using more complex sampling schemes for the locations of errors did not improve the error correction abilities. Since the notion of a synthetic error is task-specific, each task is required to implement the distribution of synthetic errors that it attempts to model for the EIFT training. To determine this distribution, we trained FT models for various base models and examined the types of errors they tended to make; see the sections below for details. We validate this overall methodology in Section A.5.

To inject a single error, we replace a golden step s_i at position i in the golden CoT with three steps according to the following procedure:

1. The erroneous step s'_i, which can easily be verified to be incorrect since we have access to the golden step s_i.

2. The error recognition step, consisting of the tokens AH! I MADE A MISTAKE.
3. Finally, the error correction step, which is just the original golden step s_i.

For EIFT training, we trained the model to predict the error recognition and error correction steps, and used loss masking to prevent the model from directly predicting the incorrect step s'_i. However, we did not modify the attention mask, as we wanted the model to learn an internal representation of what errors look like. We also ablated the token-level weight on the recognition and correction steps (Figure 7), but did not find any significant improvement in recognition or correction rates. Thus, for simplicity we did not use any special loss weighting beyond the loss masking.

A.2 MULTIPLICATION

Our multiplication task consisted of uniformly random 4 digit multiplication problems. We found that 4 digit problems were the sweet spot for task difficulty: at this complexity, the base models had low accuracy, and the FT model was significantly more accurate than the base model. Figure 1 illustrates what the different types of CoTs look like for multiplication; see Section C for more examples.

Error model. To model the errors for multiplication, we trained the FT model for `Qwen2.5` and a held out `gemma2-2b-it`. We then manually inspected the types of errors these models tended to make. Based on this study, we came up with the following broad error categories:

- `carry_error`: in an addition step, consider all locations where the column (including carry digits) sums to 9 or 10. Pick a random location and swap the column sum to the other value. Swapping 9 to 10 means incorrectly adding a carry, and swapping 10 to 9 means incorrectly omitting a carry.
- `_int_error_10`: add a random integer in $[-10, 10]$ to the answer.
- `_int_error_100`: add a random integer in $[-100, 100]$ to the answer.
- `_int_error_single_digit`: replace a randomly chosen single digit of the answer with a different random digit from 0 to 9.
- `_int_error_single_digit_close`: replace a randomly chosen single digit of the answer with a different random digit within 2 of it (respecting edge cases to keep the number syntactically valid after error injection).
- `_int_error_two_digits`: replace a random substring of two contiguous digits with two randomly chosen digits.

For the specific frequencies and a visual depiction of these error types, see Table 1 and Figure 4.

Table 1: Error probabilities for different step types. For addition steps, with probability 0.5 we inject an addition-carry error; otherwise we sample from the standard integer error injector (the `_int_error_*` distribution). For non-addition steps we always use the standard injector.

Error type	$\mathbf{Pr}[\text{error} \mid \text{addition}]$	$\mathbf{Pr}[\text{error} \mid \text{non-addition}]$
`carry_error`	0.5	0
`_int_error_10`	0.025	0.05
`_int_error_100`	0.025	0.05
`_int_error_single_digit`	0.125	0.25
`_int_error_single_digit_close`	0.125	0.25
`_int_error_two_digits`	0.20	0.40

To pick the frequencies of the errors, we hill climbed on the final accuracy for the held out model, `gemma2-2b-it`.

A.3 SUDOKU

The Sudoku task consisted of 4×4 Sudoku problems. To make the problem more tractable, we restricted to golden solutions that only make "naked single" moves.

Definition A.1 (Naked single move)**.** *For a given board state, a* naked single *is a cell that has exactly one possible candidate number remaining.*

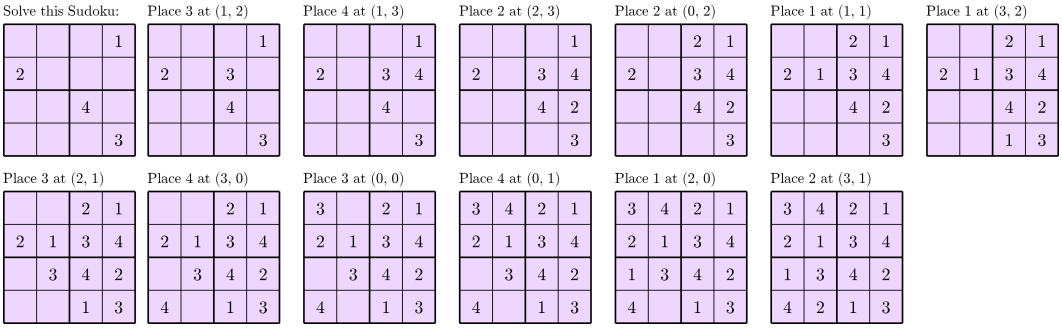


Figure 5: An example of a golden Chain-of-Thought (CoT) for the 4×4 Sudoku task. The solution trajectory is restricted to "naked single" moves, defined as instances where a specific cell has exactly one possible candidate number remaining. The problem instances are generated by solving a random board and removing numbers such that a naked single solution path remains viable.

Correspondingly, we needed to sample Sudoku problems with naked singles solutions. To do so, we first generated problems by using a Sudoku solver with randomness to generate a random valid solution. We then randomly removed numbers one by one and rejected candidates where there are no naked singles in the new board. See Figure 5 for an example of golden CoTs for sudoku.

Error model. To model errors, we used an equal split of invalid moves (i.e. moves with invalid coordinates or moves which violate a constraint) and moves that are not a naked single. We found that only injecting invalid moves did not impart error correction capabilities.

A.4 Evaluation

For evaluation, we sampled 1000 problems for each task and evaluated the FT and EIFT models using greedy sampling. We also ablated the temperature setting (across $T = 0.7, 1.0$), but did not find any significant difference in the accuracy or error correction capabilities; see Figure 12 for more discussion.

To measure the accuracy of our models, we only graded their final answers. Even at modest dataset sizes, the models are very consistent with the exact answer format that we finetuned them on. Hence, we did not observe any false positives or negatives in our grading.

To measure the error correction capabilities, we follow a more involved procedure. Of course, one should not expect the model to correct from arbitrary error distributions. To narrow the scope, we compare the error correction capabilities where the errors are sampled from the following two error distributions.

Definition A.2 (Synthetic errors)**.** *A* synthetic *error is an erroneous step in a CoT generated by our synthetic error injector.*

Definition A.3 (FT errors)**.** *An* FT *error is an erroneous step in a CoT generated by the FT model.*

In other words, we generate synthetic errors by sampling synthetic CoTs and finding errors in them, and we generate FT errors by sampling FT CoTs and finding errors in them. See Figures 8 and 9 for synthetic and FT errors, respectively, for `Qwen2.5` on multiplication.

With this in mind, we use the following process to measure error correction for the model.

1. Depending on which error source we are testing, we generate the appropriate type of CoT (synthetic or FT).
2. We truncate this CoT at the location of the first error, *including the error itself*. If the CoT does not have any errors, we discard this sample and repeat.
3. We prompt the model to complete the truncated CoT.
4. We grade the error recognition and correction for the guaranteed first error step produced by truncation. Namely, if first error step is at step i, we check for recognition on step $i + 1$ and correction on step $i + 2$. Concretely, this entails checking for the tokens AH! I MADE A MISTAKE at step $i + 1$ and checking that the step $i + 2$ is correct *as defined by our task*.

A couple points are in order about the above procedure. First, an important nuance to clarify is the exact notion of an incorrect step. To be conservative, we only check for errors that we have modeled. For multiplication, this means we only check steps where an atomic add or multiply is being performed incorrectly; for Sudoku, this means we check move placement steps and whether they are valid naked singles moves. In particular, we do not grade whether the model frivolously error corrects, as long as the proposed correction step is still correct.

Also, by convention, error corrections are a subset of error recognition—in order to correct an error, it must first declare that it has made a mistake. In practice, we never observed the model doing an error correction without first outputting an error recognition step.

A.5 Error Distributions

While we qualitatively designed the injector to mimic observed error types, it is crucial to validate if the synthetic distribution accurately covers the on-policy error distribution. Hence, we sought to quantify the alignment between the synthetic errors used during training and the natural, on-policy errors made by the models.

We measured this alignment by estimating the probability of *exact* token-level match, i.e. how often the synthetic error injector exactly replicates an FT error. First, we collected 100 reasoning traces from the FT model containing calculation errors on the multiplication task. For each erroneous instance, we truncated the trace immediately prior to the first incorrect step and sampled $n = 10,000$ candidate erroneous steps from our synthetic error injector. In comparison, during finetuning we trained the model on approximately $20,000$ incorrect steps.

Figure 6a illustrates the error coverage: the percentage of on-policy errors that appeared at least once within the n synthetic samples. We observe near-perfect coverage across all models: 95.3% for `Qwen2.5`, 99.0% for `gemma-3`, and 96.0% for `Llama-3.2`. This indicates that the support of our synthetic distribution encompasses nearly all on-policy error modes.

Further, Figure 6b demonstrates that our synthetic injector achieves distributional alignment. We plot the empirical cumulative distribution function (CDF) of the probability assigned to the exact on-policy error step among the 100 reasoning traces. In generative tasks, the space of potential incorrect tokens is vast, making any significant probability mass on an exact string match a strong indicator of alignment. We find that our injector frequently assigns high probabilities to the model's specific natural errors. For instance, it assigns greater than 15% probability to the exact on-policy errors made by `Qwen2.5` and `gemma-3` in a large fraction of CoTs (70% for `Qwen2.5` and 30% for `gemma-3`). Hence, during training, the models frequently encounter the *exact* errors they are prone to making, confirming the quality of the synthetic error distribution.

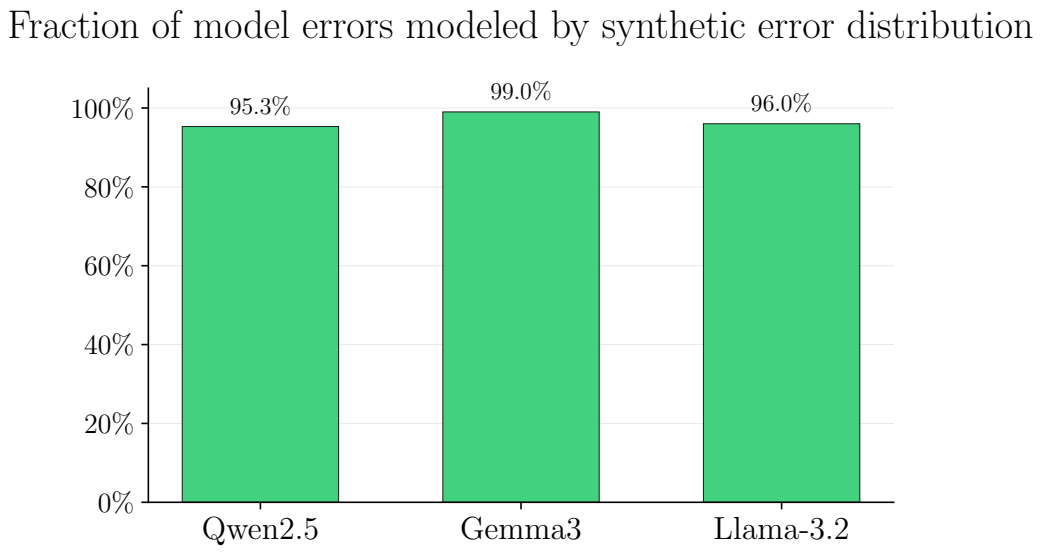
In Tables 2 and 3, we analyze the input FT error correction capabilities for `Qwen2.5`, bucketed by the synthetic error categories. Perhaps unsurprisingly, we do not exactly match the frequencies of error categories. More notably, the correction rates conditioned to specific error buckets were significantly lower for FT errors. Indeed, manual inspection of the FT errors revealed that they tended to be more subtle to catch and fix than synthetic errors: the relative error of FT errors was on average an order of magnitude smaller.

B Training details

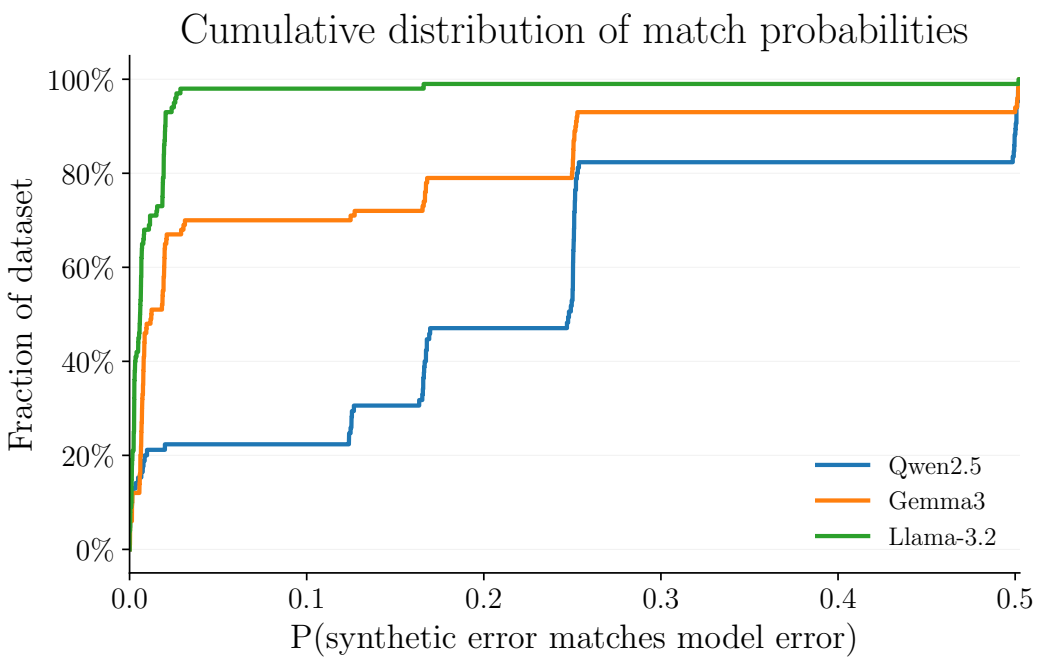
Each model was trained on a single GPU using `bfloat16` mixed precision for 10000 steps with a batch size of 4 and a max sequence length of 1024 (the typical prompt length depended on the task). We used the AdamW optimizer with learning rate `2e-5`, weight decay `1e-2`, and default hyperparameters otherwise.

In Figure 7 we present our ablation over the token-level weight for the correction and backtrack steps.

C CoT examples



(a)



(b)

Figure 6: **Validation of synthetic error distribution alignment with on-policy errors. (a) Error Coverage:** The percentage of on-policy model errors that appear at least once within $n = 10,000$ samples from our synthetic error injector. We achieve near-perfect coverage ($> 95\%$) across all models, indicating the injector correctly identifies the support of the error distribution. **(b) Distributional Alignment:** The cumulative distribution of the probability assigned by the synthetic injector to the *exact* on-policy error step (down and to the right is better). For a given probability threshold p on the x-axis, the y-axis represents what fraction of the 100 errors could be exactly matched with probability at most p. The probability mass placed on exact matches (particularly for `Qwen2.5` and `gemma-3`) indicates that the synthetic distribution closely approximates the frequency of natural model errors.

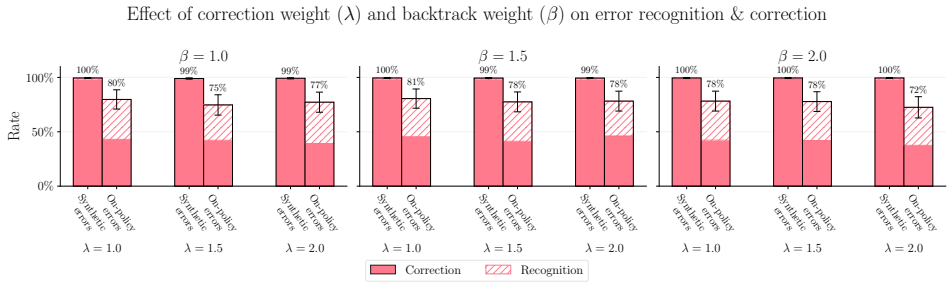


Figure 7: Sweep of correction weight λ and backtrack weight β across $\{1.0, 1.5, 2.0\} \times \{1.0, 1.5, 2.0\}$ for the tokens in the error correction step for `Qwen2.5` on multiplication. The error recognition and correction rates remain relatively consistent across the entire sweep.

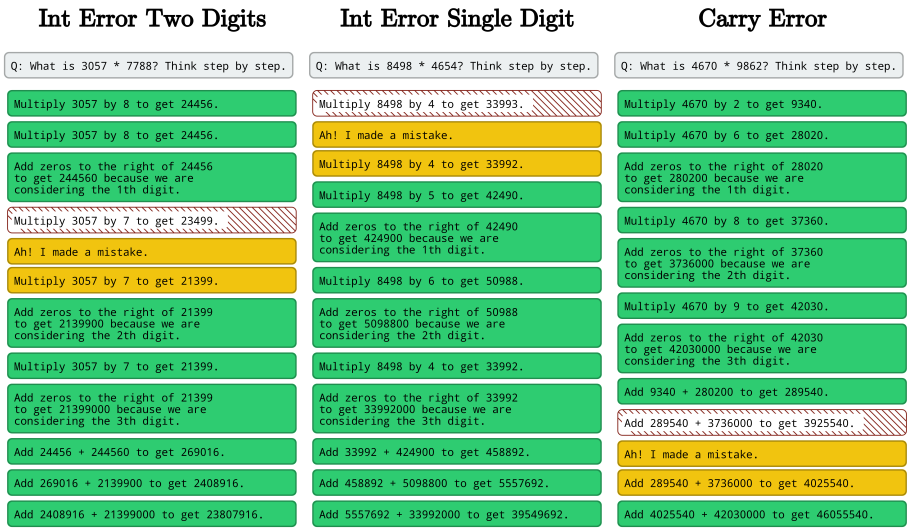


Figure 8: Representative examples of synthetic errors for multiplication. The hatched red steps are the incorrect steps, and the yellow steps are the error recognition/correction steps. See Section A.2 for more details on what these types of errors entail.

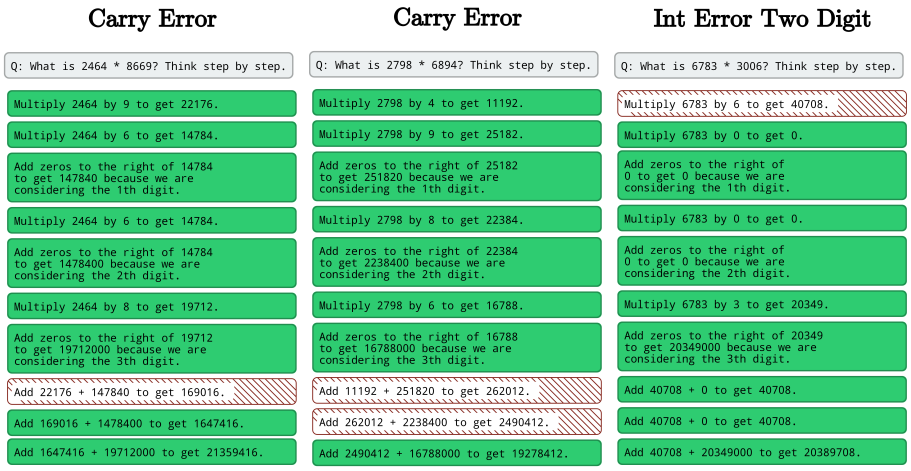


Figure 9: Representative examples of FT errors for multiplication from `Qwen2.5`. The hatched red steps are the incorrect steps. See Section A.2 for more details on what these types of errors entail.

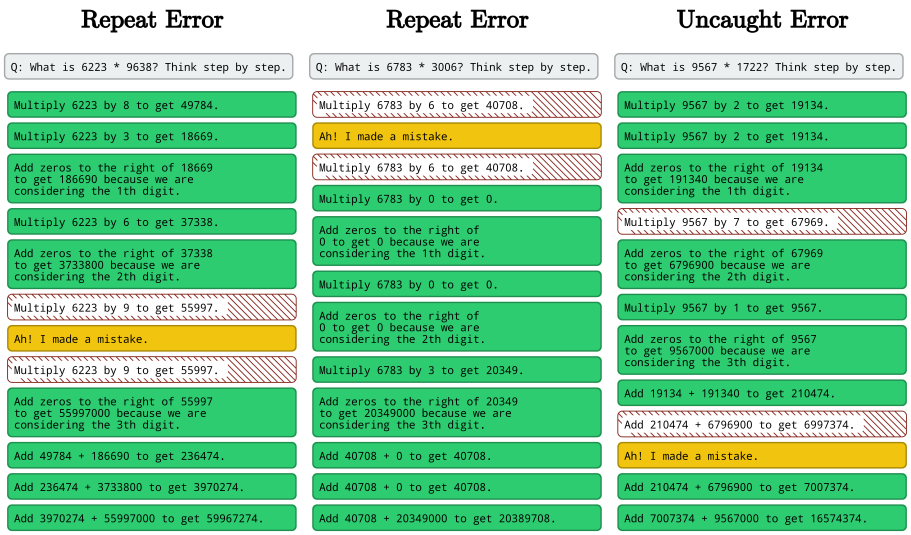


Figure 10: Representative examples of EIFT self-correction failure modes from `Qwen2.5`. The hatched red steps are incorrect, and the yellow steps are the recognition steps. In the (Left) and (Center) panels we see instances where the EIFT model mimics its original mistake after recognizing it. In the (Right) panel we see a failure to recognize a mistake, along with a successful error correction near the end of the CoT.

Table 2: Error recognition and correction breakdown for the `Qwen2.5` synthetic EIFT model on 1000 synthetic errors. Note that there is some overlap in error categories; we greedily classified in the order `carry_error` → `_int_error_single_digit_close` → `_int_error_single_digit` → `_int_error_two_digits` → `_int_error_10` → `_int_error_100`.

Input Error Type	Frequency	Recognition Rate	Correction Rate	Correction | Recognition
`_int_error_10`	0.002	1.000	1.000	1.000
`_int_error_100`	0.012	1.000	1.000	1.000
`_int_error_single_digit`	0.215	0.986	0.981	0.995
`_int_error_single_digit_close`	0.287	0.997	0.990	0.993
`_int_error_two_digits`	0.348	1.000	0.986	0.986
`carry_error`	0.136	0.993	0.985	0.993

Table 3: Error recognition and correction breakdown for the `Qwen2.5` synthetic EIFT model on 1000 FT errors. The classification order is the same as in Table 2.

Input Error Type	Frequency	Recognition Rate	Correction Rate	Correction | Recognition
`_int_error_10`	0.042	1.000	0.429	0.429
`_int_error_100`	0.054	1.000	0.574	0.574
`_int_error_single_digit_close`	0.052	0.058	0.058	1.000
`_int_error_two_digits`	0.069	0.899	0.449	0.500
`carry_error`	0.733	0.810	0.465	0.574
`unknown`	0.050	0.780	0.420	0.538

Solve this Sudoku:

	1		
		2	
			4
4			

Place 3 at (0, 3)

	1		3
		2	
			4
4			

Place 3 at (1, 0)

	1		3
3		2	
			4
4			

Place 2 at (0, 0)

2	1		3
3		2	
			4
4			

Place 4 at (0, 2)

2	1	4	3
3		2	
			4
4			

Place 4 at (1, 1)

2	1	4	3
3	4	2	
			4
4			

Place 1 at (1, 3)

2	1	4	3
3	4	2	1
			4
4			

Place 1 at (2, 0)

2	1	4	3
3	4	2	1
1			4
4			

Place 2 at (3, 3)

2	1	4	3
3	4	2	1
1			4
4			2

Place 3 at (2, 2)

2	1	4	3
3	4	2	1
1		3	4
4			2

Place 3 at (3, 1)

2	1	4	3
3	4	2	1
1		3	4
4	3		2

Place 2 at (2, 1)

2	1	4	3
3	4	2	1
1	2	3	4
4	3		2

Place 1 at (3, 2)

2	1	4	3
3	4	2	1
1	2	3	4
4	3	1	2

Solve this Sudoku:

		2	
	4		
4	3		

Place 1 at (0, 1)

	1	2	
	4		
4	3		

Place 1 at (3, 2)

	1	2	
	4		
4	3	1	

Place 3 at (0, 0)

3	1	2	
	4		
4	3	1	

Place 3 at (1, 2)

3	1	2	
	4	3	
4	3	1	

Place 2 at (2, 1)

3	1	2	
	4	3	
	2		
4	3	1	

Place 2 at (3, 3)

3	1	2	
	4	3	
	2		
4	3	1	2

Place 4 at (0, 3)

3	1	2	4
	4	3	
	2		
4	3	1	2

Place 2 at (1, 0)

3	1	2	4
2	4	3	
	2		
4	3	1	2

Place 1 at (1, 3)

3	1	2	4
2	4	3	1
	2		
4	3	1	2

Place 1 at (2, 0)

3	1	2	4
2	4	3	1
1	2		
4	3	1	2

Place 4 at (2, 2)

3	1	2	4
2	4	3	1
1	2	4	
4	3	1	2

Place 3 at (2, 3)

3	1	2	4
2	4	3	1
1	2	4	3
4	3	1	2

Solve this Sudoku:

	3		
4			
	2	3	

Place 1 at (1, 1)

	3		
4	1		
	2	3	

Place 1 at (3, 0)

	3		
4	1		
1	2	3	

Place 2 at (0, 0)

2	3		
4	1		
1	2	3	

Place 2 at (1, 2)

2	3		
4	1	2	
1	2	3	

Place 3 at (2, 0)

2	3		
4	1	2	
3			
1	2	3	

Place 4 at (2, 1)

2	3		
4	1	2	
3	4		
1	2	3	

Place 4 at (3, 3)

2	3		
4	1	2	
3	4		
1	2	3	4

Place 1 at (0, 3)

2	3		1
4	1	2	
3	4		
1	2	3	4

Place 3 at (1, 3)

2	3		1
4	1	2	3
3	4		
1	2	3	4

Place 1 at (2, 2)

2	3		1
4	1	2	3
3	4	1	
1	2	3	4

Place 4 at (0, 2)

2	3	4	1
4	1	2	3
3	4	1	
1	2	3	4

Place 2 at (2, 3)

2	3	4	1
4	1	2	3
3	4	1	2
1	2	3	4

Figure 11: Representative examples of golden chain-of-thoughts (CoTs) from the sudoku task.

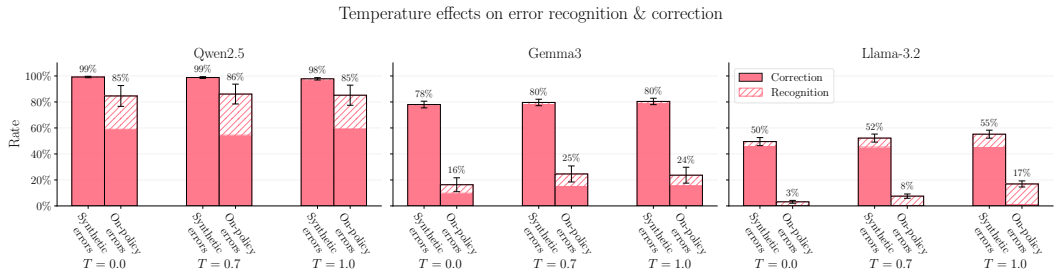


Figure 12: Temperature ablation for multiplication evaluation. We compare the correction and recognition rates for the exact same multiplication checkpoints, using different temperature settings: $T \in \{0.0, 0.7, 1.0\}$. We find no significant change in the capabilities across temperature settings for Qwen. For Gemma and Llama, we only see modest gains in performance with positive temperature, but the gap between synthetic and on-policy performance remains large ($> 40\%$ absolute difference).