On-Policy Distillation

Kevin Lu in collaboration with others at Thinking Machines

Oct 27, 2025

LLMs are capable of expert performance in focused domains, a result of several capabilities stacked together: perception of input, knowledge retrieval, plan selection, and reliable execution. This requires a stack of training approaches, which we can divide into three broad stages:

- **Pre-training** teaches general capacities such as language use, broad reasoning, and world knowledge.
- **Mid-training** imparts domain knowledge, such as code, medical databases, or internal company documents.
- **Post-training** elicits targeted behavior, such as instruction following, reasoning through math problems, or chat.

Smaller models with stronger training often outperform larger, generalist models in their trained domains of expertise. There are many benefits to using smaller models: they can be deployed locally for privacy or security considerations, can continuously train and get updated more easily, and save on inference costs. Taking advantage of these requires picking the right approach for the later stages of training.

Approaches to post-training a "student" model can be divided into two kinds:

- **On-policy training** samples rollouts from the student model itself, and assigns them some reward.
- **Off-policy training** relies on target outputs from some external source that the student learns to imitate.

For example, we may wish to train a compact model to solve math questions such as:

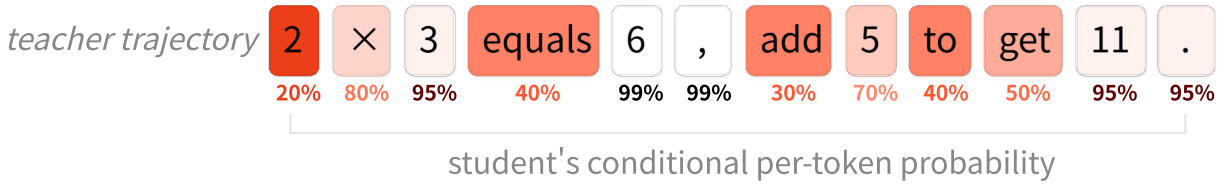
We can do on-policy training via reinforcement learning, by grading each student rollout on whether it solves the question. This grading can be done by a human, or by a "teacher" model that reliably gets the correct answer.

The strength of on-policy training is that by training on samples from itself, the student learns to avoid mistakes in a more direct way. But RL has a major downside: it provides very sparse feedback, teaching a fixed number of bits per training episode regardless of the number of tokens used. In our example above, the student learns that "21" is the wrong answer and updates away from producing the rollout it tried. But it doesn't learn where exactly the mistake was made, whether it got the order of operations wrong or erred in the arithmetic itself. This sparsity of feedback makes RL inefficient for many applications.

Off-policy training is often done with supervised fine-tuning (SFT): training on a curated set of task-specific labeled examples. The source of these labeled examples can be a teacher model that is proven to perform well on the task at hand.

We can use a mechanism called **distillation:** training the student to match the output distribution of a teacher model. We train on teacher **trajectories:** the complete sequence of generated tokens including intermediate thinking steps. We can use the teacher's full next-token distribution at each step (often called "logit distillation") or just sample given sequences. In practice, sampling sequences provides an unbiased estimation of the teacher's distribution and arrives at the same

objective. The student updates towards each token in the sequence in proportion to how unlikely it was to generate that token itself, represented by darker color in the example below:



Distillation from large model teachers has proven effective in training small models to follow instructions,[1] reason on math and science,[2] extract clinical information[3] from medical notes, and engage in multi-turn chat dialogues.[4] The distillation datasets used for these and other applications are often open-sourced and published.

The drawback of off-policy training is that the student learns in contexts frequented by the teachers, not ones the student itself will often find itself in. This can cause compounding error: if the student makes an early mistake that the teacher never makes, it finds itself diverging ever farther from the states it observed in training. This problem becomes particularly acute when we care about the student's performance on long sequences. To avoid this divergence, the student must learn to recover from its own mistakes.

Another issue observed with off-policy distillation is that the student can learn to imitate the teacher's style and confidence but not necessarily its factual accuracy.[5]

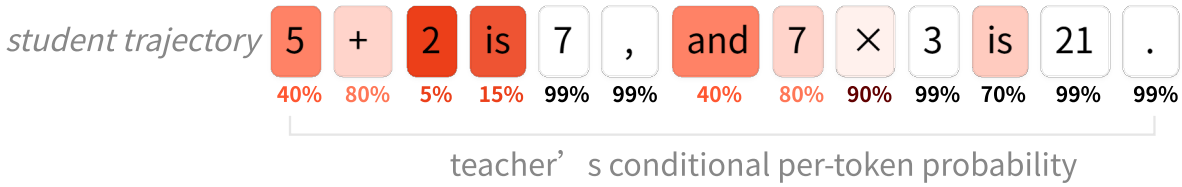
If you're learning to play chess, on-policy RL is analogous to playing games with no coaching. The feedback of winning or losing a match is tied directly to your own play, but is received only once per match and doesn't tell you which moves contributed most to the outcome. Off-policy distillation is analogous to watching a grandmaster playing — you observe extremely strong chess moves, but they are played in board states that a novice player will rarely find themselves in.

We want to combine the on-policy relevance of RL with the dense reward signal of distillation. For learning chess, this would be a teacher that grades each of *your own* moves on a scale from "blunder" to "brilliant". For LLM post-training, it's on-policy distillation.

Figure 4: Screenshot from chess.com. Each move is color-graded by an analysis engine, which labels moves as blunders (red), mistakes (orange), inaccuracies (yellow), or brilliant (blue).

On-policy distillation — best of both worlds

The core idea of on-policy distillation is to sample trajectories from the *student* model and use a high-performing teacher to grade *each token* of each trajectory. Returning to our math example above, on-policy distillation would score each step of the solution punishing the mistakes that caused the student to arrive at the wrong answer while reinforcing the ones that were executed correctly.



In this post, we explore the application of on-policy distillation for tasks such as training a model for math reasoning and training an assistant model that combines domain knowledge with instruction following. We apply on-policy distillation on models that have a foundation of capabilities from pre- and mid-training. We find that it is a cheap and powerful approach to post-training, combining the advantages of on-policy training with a dense reward signal.

Method	Sampling	Reward signal
Supervised finetuning	off-policy	**dense**
Reinforcement learning	**on-policy**	sparse
On-policy distillation	**on-policy**	**dense**

Our work with on-policy distillation draws inspiration from DAGGER,[6] an iterative SFT algorithm that includes teacher evaluations of student-visited states. It is also similar to process reward modeling,[7] an RL approach that scores every step in the student model's chain-of-thought. We extend prior on-policy distillation work by Agarwal et al,[8] Gu et al,[9] and the Qwen3 team[10]. Using the Tinker training API, we replicate Qwen3's result of achieving equivalent performance on reasoning benchmarks with on-policy distillation for a fraction the cost of RL.

Implementation

You can follow along with each step of the implementation in the Tinker cookbook.

Loss function: reverse KL

On-policy distillation can use a variety of loss functions for grading the student's trajectories.[11] For simplicity, we choose the per-token reverse KL — the divergence between the student's (π_θ) and teacher's (π_{teacher}) distribution for each token conditioned on the same prior trajectory:

$$\mathrm{KL}\left(\pi_\theta || \pi_{\text{teacher}}\right) = \mathbb{E}_{x\sim\pi_\theta}\left[\log \pi_\theta(x_{t+1}|x_{1..t}) - \log \pi_{\text{teacher}}(x_{t+1}|x_{1..t})\right]$$

Our reward function minimizes the reverse KL, which pushes the student to approximate the teacher's behavior in every state the student finds itself in. When the student behaves identically to the teacher, reverse KL is zero. For simplicity, we use a discount factor of zero: at any given timestep,[12] the student only optimizes the immediate next token, with no consideration for future tokens.

Reverse KL has natural synergy with RL, which generally optimizes a form of sequence-level reverse KL induced by the reward model. However, unlike most reward models in practice, the reverse KL is "unhackable" in the sense that low KL always corresponds to a high probability of desirable behavior from the teacher model's point of view. Two other useful properties of reverse KL are that it is "mode seeking"[13] — it learns one specific behavior (the teacher's) instead of spreading its distribution across several suboptimal options — and it reduces exposure bias.[14]

This approach offers significant compute savings. Since it doesn't require a rollout to finish sampling to calculate the reward, we can use shorter or partial rollouts for training. Querying the teacher's log probabilities also requires just a single forward pass from the larger model, while the trajectories are generated by the smaller and cheaper student.

We do not require a separate reward or labeling model either. There could be advantages to combining distillation-based per-token rewards with sequence-level environment rewards; this is an interesting area for potential future research.

Illustration

Below we see a real example of an incorrect student trajectory graded by a teacher. The example is from SimpleBench, and relies on the model making a key observation that the question's premise is important: the correct answer is "B. 0" since ice cubes will melt in a frying pan. The student, Qwen3-4B-Instruct-2507, incorrectly treats this as a pure math problem with no consideration of the physical context.

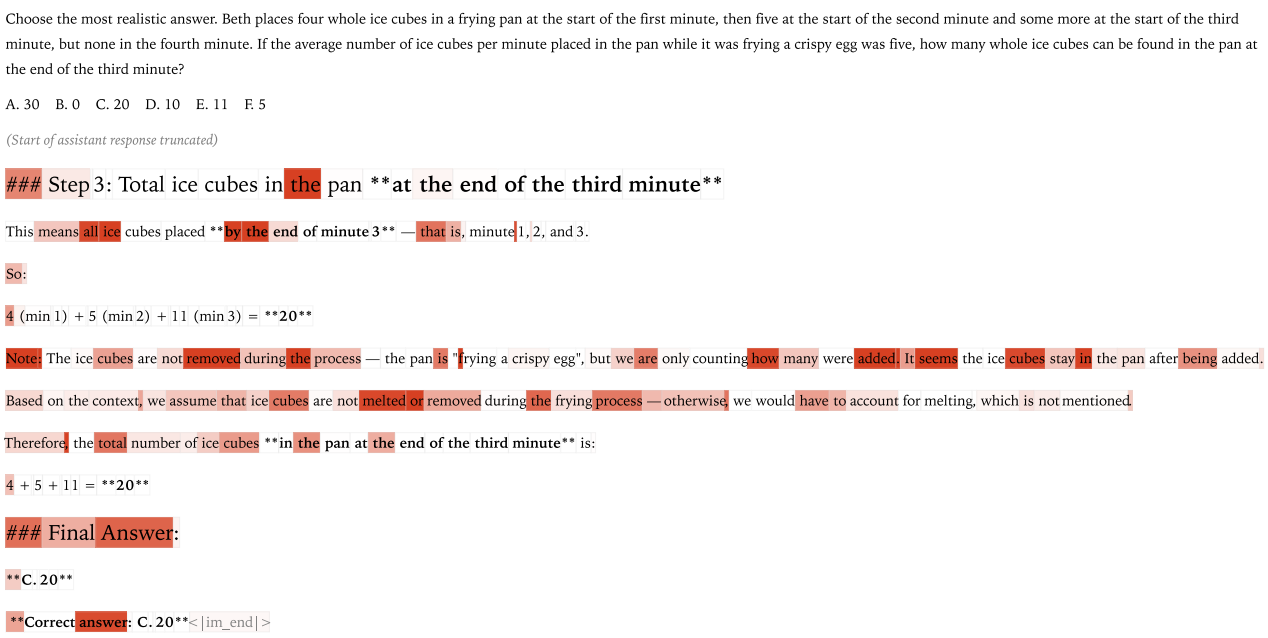


Figure 6: Example trajectory graded by a teacher model. Tokens in darker red correspond to a higher reverse KL.

Darker colors represent tokens that receive a higher penalty from the teacher model, Qwen3-235B-A22B-Instruct-2507, which solves this problem correctly. We see that it penalizes tokens that start phrases which lead the student astray, intuitively corresponding to important "forking tokens" for guiding reasoning.[15] The final answer, though wrong, isn't penalized — it is entirely predictable conditional on the whole preceding sequence.

Pseudocode

We implement on-policy distillation on top of the RL script in Tinker, which already implements sampling, reward computation, and policy gradient-style training.[16]

1. *Initialize teacher client.* The Tinker API enables easily creating different clients for different models, without needing to worry about the utilization of model engines. We use a sampling client, as we do not need to propagate logprobs through the teacher model.
2. *Sample trajectories.* We sample rollouts from the student exactly as we would in RL. During sampling, RL already computes the student's logprobs $\log \pi_\theta(x)$ for use as part of the importance-sampling loss.
3. *Compute reward.* We query the teacher client with `compute_logprobs` on the sampled trajectories, which returns the teacher's logprobs $\log \pi_{\text{teacher}}(x)$ on the tokens x sampled by the student.[17] We then use this to calculate the reverse KL.
4. *Train with RL.* We set the per-token advantage to the negative reverse KL, and call the RL importance-sampling loss function to perform the training update on the student model.

python

```
# Initialize teacher client (main):
teacher_client = service_client.create_sampling_client(
    base_model=teacher_config.base_model,
    model_path=teacher_config.load_checkpoint_path,
)

# Sample trajectories (main):
trajectories = do_group_rollout(student_client, env_group_builder)
sampled_logprobs = trajectories.loss_fn_inputs["logprobs"]

# Compute reward (compute_teacher_reverse_kl):
teacher_logprobs = teacher_client.compute_logprobs(trajectories)
reverse_kl = sampled_logprobs - teacher_logprobs
trajectories["advantages"] = -reverse_kl

# Train with RL (train_step):
training_client.forward_backward(trajectories, loss_fn="importance_sampling")
```

In the experiments below, we generally apply on-policy distillation to models that have already been mid-trained on specific domain knowledge. This training increases the probability that the student will generate tokens within the teacher's distribution, though it's usually far from sufficient for replicating the teacher's performance.[18] Often, as we'll see in our personalization example, the probability of generating the relevant tokens starts at zero as the student lacks any relevant domain knowledge.

We use on-policy distillation for post-training, and compare it to other approaches to this last crucial stage of training expert models.

Distillation for reasoning

We use distillation to train mathematical reasoning in the Qwen3-8B-Base model, using Qwen3-32B as a teacher model. Both the teacher (Qwen3-32B) and student (Qwen3-8B-Base) are supported models on Tinker today, so you can reproduce our experiments with the Tinker cookbook.

Off-policy distillation

As mentioned above, all our experiments start with mid-training in the form of off-policy distillation — supervised fine-tuning on a dataset of teacher-generated examples. The dataset used for mathematical reasoning is OpenThoughts-3, a collection of reasoning prompts and responses generated by QwQ-32B (a reasoning model similar to Qwen3-32B).

Training the student (Qwen3-8B-Base) on 400k prompts with full fine-tuning achieves a score of 60% on AIME'24, a benchmark of math problems. We can also train with LoRA,[19] though it lags behind full fine-tuning when training on high-volume datasets. In all cases, we see performance increase log-linearly — the initial performance gains are cheap but the latter are costly.

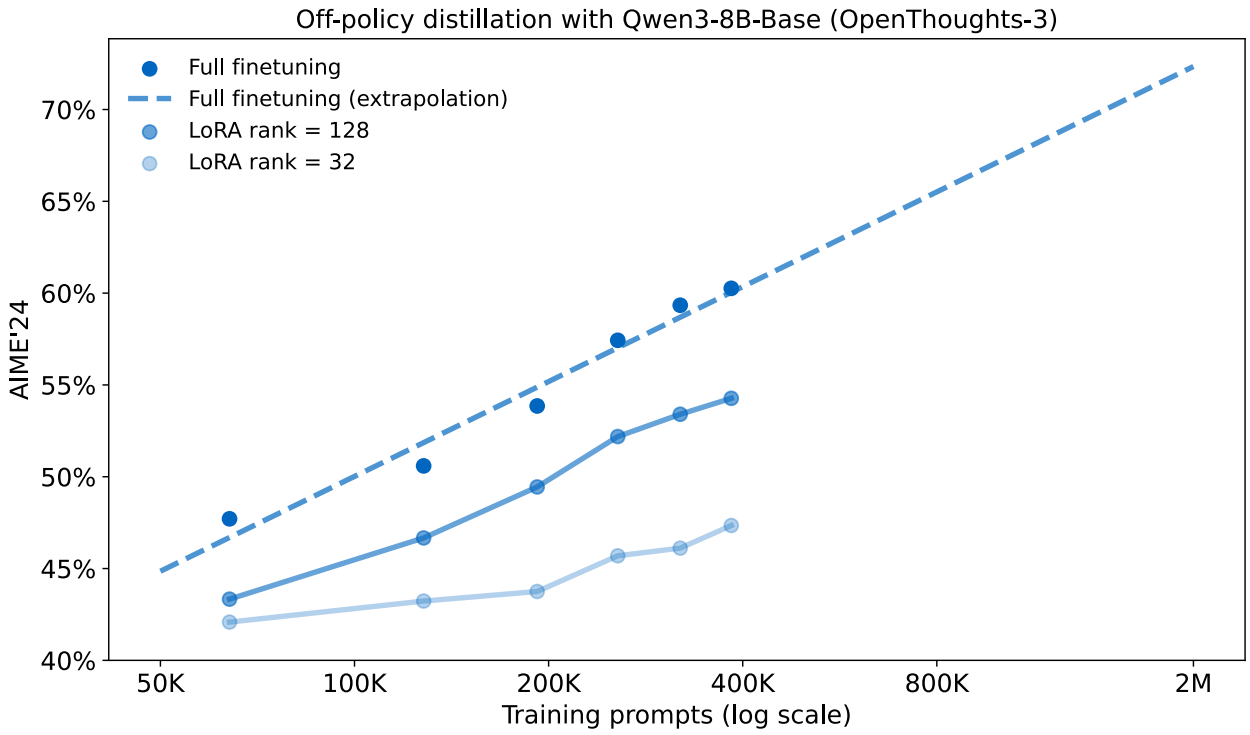


Figure 7: AIME'24 score over the course of off-policy distillation (SFT). After the initial 50-100K prompts, performance follows a predictable log-linear scaling curve. As predicted in LoRA Without

Regret, we observe worse LoRA performance when running large-scale SFT with high batch size.

We can treat the model fine-tuned on 400k prompts as a checkpoint before trying various post-training approaches to increase its performance. We can compare the effort it would take to raise the score on the AIME'24 benchmark from 60% to 70%.

The default approach is to fine-tune on more prompts, continuing the process of off-policy distillation. Extrapolating the log-linear trend, we estimate that the model would achieve 70% on AIME'24 at approximately 2M prompts. This extrapolation requires the scaling law to hold up without stalling, which isn't trivial. However, there are examples of large-scale off-policy distillation improving an 8B model's performance past 70%, such as OpenThoughts-3 and DeepSeek-R1-0528-Qwen3-8B.[20] We can use this extrapolation as an optimistic estimate for the cost-performance ratio of off-policy distillation.

Reinforcement learning

The Qwen3 technical report reaches performance of 67.6% on the benchmark using 17,920 GPU hours of RL on top of a similar SFT initialization. It is hard to compare this directly to the cost of distillation, but given some reasonable assumptions about the SFT training stack, this is similar to the cost of training on 2M off-policy distillation prompts.

Method	AIME'24	GPQA-Diamond	GPU Hours
Off-policy distillation	55.0%	55.6%	Unreported
+ Reinforcement learning	67.6%	61.3%	17,920
+ On-policy distillation	**74.4%**	**63.3%**	**1,800**

From Qwen3 Technical Report, Table 21.

The Qwen team also reports reaching a higher score of 74.4 on AIME'24 at one-tenth the cost of RL with on-policy distillation, which served as inspiration for our work. We attempt to replicate it below in our basic setup.

On-policy distillation

As an alternative to off-policy distillation or RL, we run on-policy distillation as described above.[21] Starting from the 400k SFT checkpoint, on-policy distillation achieves an AIME'24 of 70% in about 150 steps.[22]

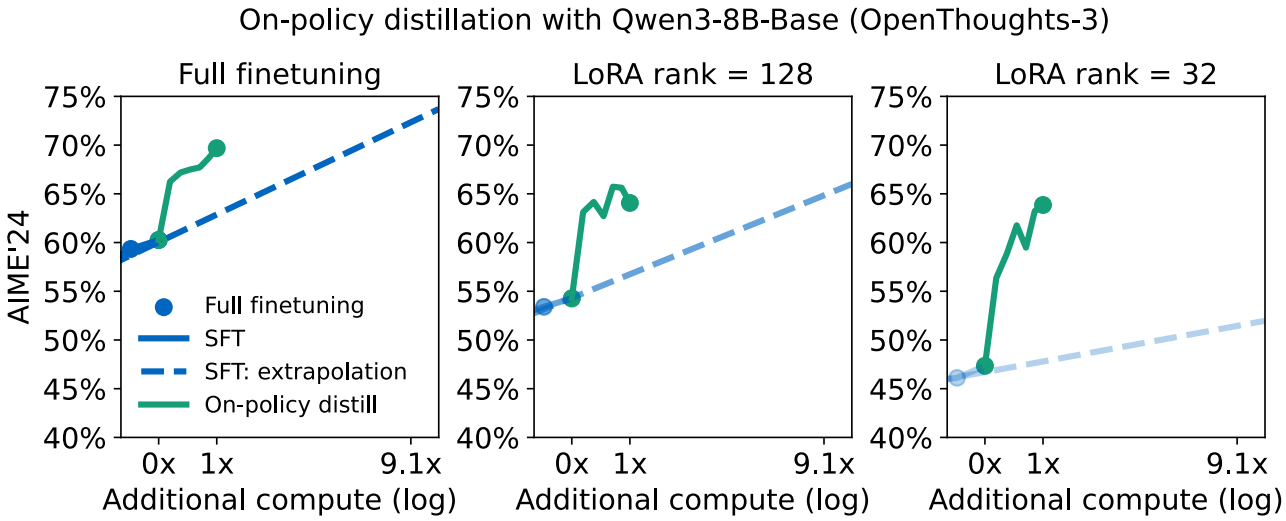


Figure 8: AIME'24 over the course of on-policy distillation. We measure additonal compute in terms of training FLOPs (see below). On-policy distillation is significantly more compute-efficient than SFT, especially for LoRA models. At rank = 32, LoRA trails full finetuning by 13% after SFT, but only 6% after on-policy distillation.

Comparing compute costs across methods is nontrivial, as the ratio of training vs sampling vs log-prob computation cost varies significantly depending on implementation. Below, we compute the cost in terms of FLOPs which penalizes methods that can be effectively parallelized on GPUs. In particular, it overestimates the practical cost of computing log-probs.

Method	AIME'24	Teacher FLOPs	Student FLOPs	CE vs SFT-2M
Initialization: SFT-400K	60%	8.5×10^{20}	3.8×10^{20}	–
SFT-2M (extrapolated)	~70% (extrapolated)	3.4×10^{21}	1.5×10^{21}	1×
Reinforcement learning	68%	-	-	≈1×
On-policy distillation	70%	8.4×10^{19}	8.2×10^{19}	**9-30×**

We find a baseline cost reduction of 9x when the SFT dataset is given, as in our example with OpenThoughts-3, or is amortized across many training runs.[23] In this case we do not count the cost of teacher FLOPs for off-policy training but do for on-policy, since we must run the teacher model to compute log-probs for the student's trajectory. Since this computation can be cheaply parallelized across GPUs, the cost reduction in GPU hours is closer to 18x.

However, we often want to train a small model on a new task for which no off-policy distillation dataset is available. If we include the full cost of the teacher model in off-policy distillation – ie, including the additional cost of sampling from the teacher model – the total cost reduction is about 30x.[24]

Distillation for personalization

In addition to training small models to high performance on common tasks, another use-case for distillation is personalization. Examples include adhering to a particular tone in conversation and format of output, or capabilities like tool use and cost budgeting. We often want to train this behavior in combination with new domain knowledge.

Training both at once is generally difficult, and light-weight finetunes are often insufficient for this objective,[25] thereby requiring a larger midtrain. Learning post-training behaviors on top of new knowledge requires a complex post-training stack, often consisting of proprietary data and reward models. While this approach is accessible to frontier labs, it can be difficult or prohibitively expensive for other practitioners to replicate.

In this section, we show that on-policy distillation can be used effectively to post-train specialized behavior. This approach is also applicable to continual learning or "test-time training": updating models when they are deployed without regressing the base performance. We use an example application of a model mid-trained on our internal company documents.

Training an internal assistant

A common objective for a custom model is to act as an assistant: to possess expert knowledge in some field and, in addition, reliable assistant-like behavior. We may need separate training for each, especially when the domain of expertise can't be learned from pre-training data only or when learning it interferes with behavior.

Our example is an internal company assistant, for which we have two desiderata:

1. The model is **knowledgeable** about the domain (company documents). Pretrained models have seen zero internal documents from the company, and therefore can only guess, regardless of model scale. We will measure this using an internal knowledge recall eval ("internal QA").
2. The model exhibits strong **post-training** behavior, ie. instruction following. We will measure this with the commonly-used IF-eval.[26]

Training on new knowledge degrades learned behavior

We will start with Qwen3-8B, rather than the base model. Qwen3-8B is post-trained on useful skills for an assistant, such as instruction following and reasoning with RL. Prior research has shown that such reinforcement learning only trains small subnetworks of the original model,[27] and thus can be fragile when the network is further trained on a large amount of data. We study the extent to which it happens, and how the desired behavior can be recovered.

In order to reduce such catastrophic forgetting, a common approach in mid-training is to mix in "background data" from the original model's pretraining distribution.[28] In this case, we don't have access to Qwen3's pretraining distribution. Therefore, we consider a stronger and more expensive baseline: we take Tulu3[29] prompts – a broad chat and instruction-following dataset – and re-sample them with Qwen3-8B in order to act as chat background data.

This "on-policy" background data sampled by Qwen3-8B acts as a forwards KL regularizer, reinforcing the model's original behavior throughout mid-training. We find sampling from Qwen3-8B is better than Qwen3-32B for preserving chat capabilities throughout mid-training, highlighting the sensitivity of the data source; similar on-policy SFT results have been found in Chen et al.[30] We hypothesize this approach can be even more effective than having access to the original pretraining data distribution, at the cost of having to sample a large-scale dataset.

We then fine-tune Qwen3-8B on different mixes of our internal documents and chat data. Increasing the proportion of document data straightforwardly improves the model's knowledge. However, although mixing in at least 30% of chat data helps preserve most instruction-following ability, there is no weighting which maintains the original performance on IF-eval.[31]

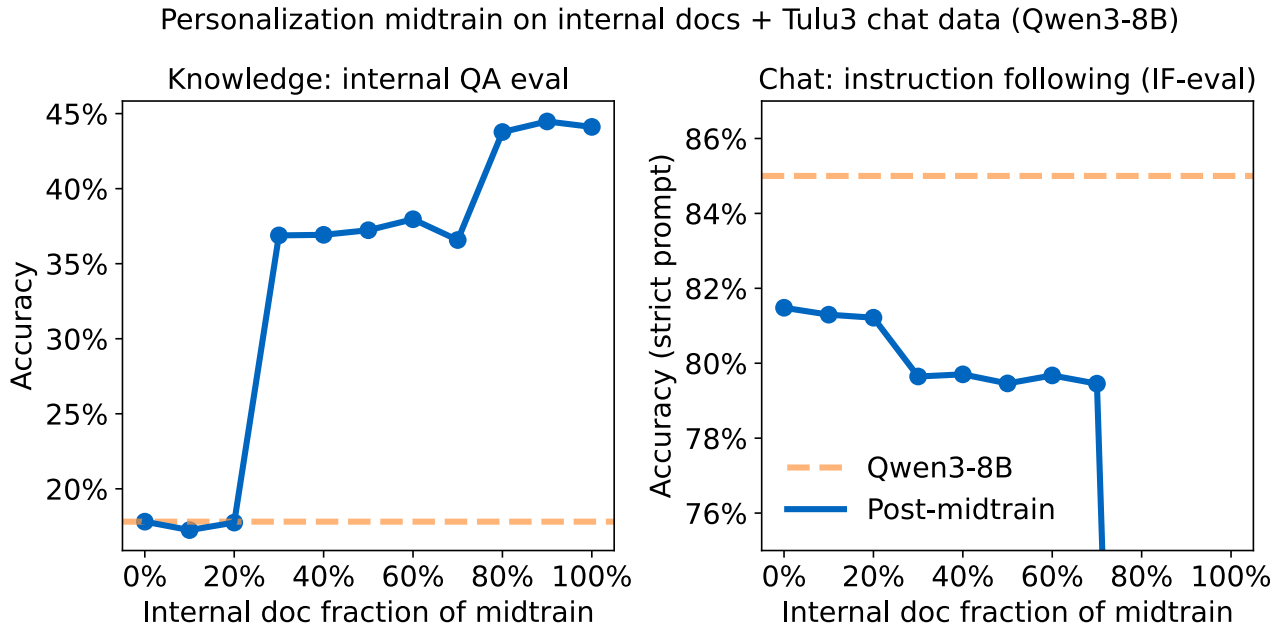


Figure 9: Sweeping over the ratio of internal documents: background chat data during mid-training. Although mixing in a little chat data helps to prevent a catastrophic regression, no weight maintains the original IF-eval performance.

For any given mix, we observe IF-eval performance degrades during fine-tuning. This compromises our ability to use longer training to help specialize the model further.[32]

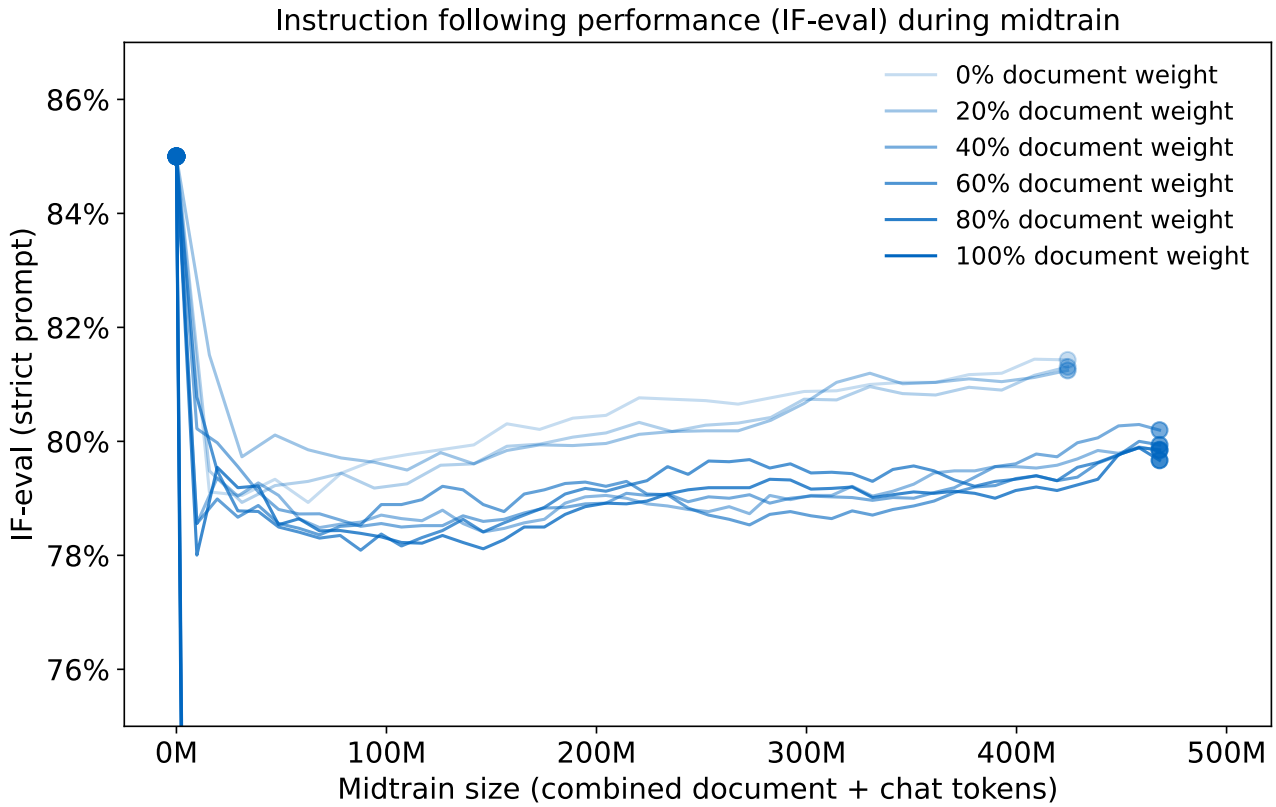


Figure 10: IF-eval decreases during midtraining across all mixes of data. When we use a linear learning rate (pictured), the degradation eventually flattens and slowly begins to recover as the learning rate decays. However, performance never fully recovers.

An alternative commonly-used approach is to use LoRA in order to constrain the parameter update, thereby reducing the possibility of catastrophic forgetting.

However, this approach is still insufficient for preserving IF-eval, and the LoRA learns less.[33]

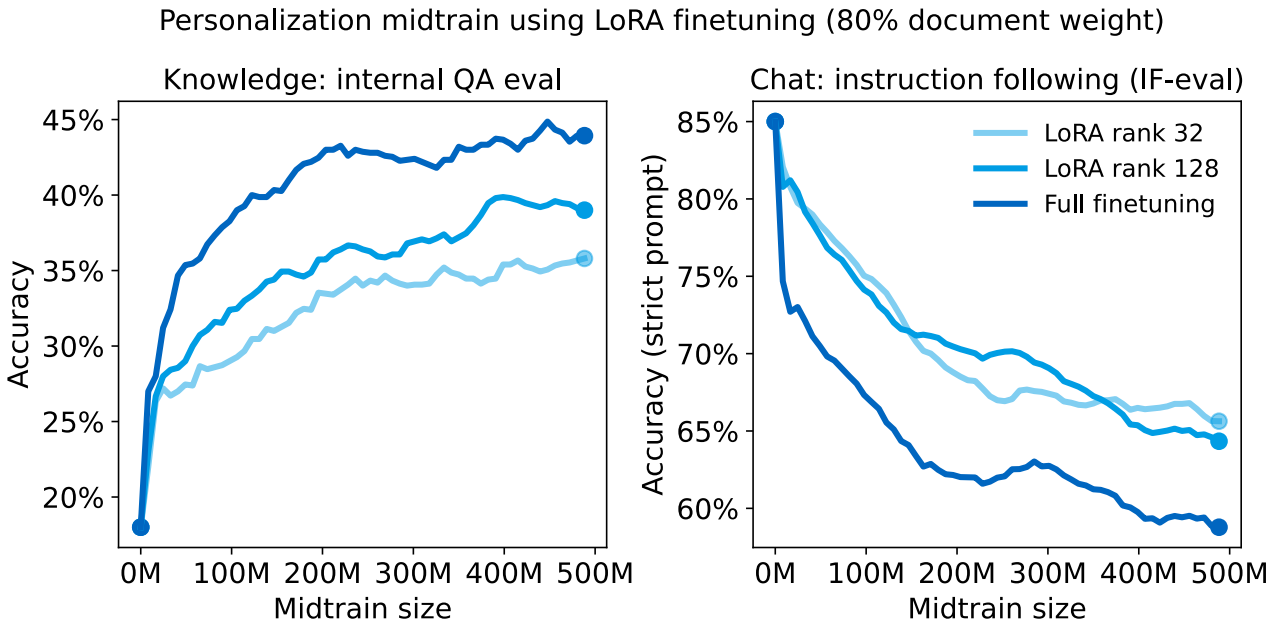


Figure 11: When used for our personalization midtrain on top of the post-trained Qwen3-8B, LoRA learns less (knowledge) and still forgets its original post-training behaviors.

On-policy distillation recovers post-training behavior

Next, we seek to restore instruction-following behavior after fine-tuning on internal documents. This behavior was originally trained with RL which is expensive and, as we have seen, fragile. Instead, we run on-policy distillation with the earlier version of the model, Qwen3-8B, as the teacher on Tulu3 prompts. Note that this phase of training has no relation to the internal document data, and is designed solely to recover instruction following.

The use of an earlier version of the model as a teacher to "re-invoke" capabilities lost during fine-tuning makes on-policy distillation very promising for continuous learning. We could alternate between phases of fine-tuning on new data and distillation to recover behavior to allow our model to learn and stay up-to-date on knowledge over time. This phase-alternating approach has previously been explored by Cobbe et al.[34]

After fine-tuning on an 70-30 mix of internal document data and chat data, on-policy distillation recovers nearly full performance on IF-eval without losing any knowledge; we also observe some positive transfer between chat capabilities and the model's "knowledge" performance on the internal QA eval.

Model	Internal QA Eval (Knowledge)	IF-eval (Chat)
Qwen3-8B	18%	**85%**
+ midtrain (100%)	**43%**	45%
+ midtrain (70%)	36%	79%
+ midtrain (70%) + distill	**41%**	**83%**

Domain-specific (Internal QA eval) and chat (IF-eval) performance after mid-training. Although mid-training forgets the post-trained behaviors of Qwen3-8B, they are cheaply restored via on-policy distillation, alongside the additional knowledge learned via the mid-train.

In essence, we have treated the language model itself as a reward model, with high-probability behaviors being rewarded.[35] This has connections to inverse RL: high-probability behaviors correspond to advantageous rewards in an assumed underlying preference model.[36] Any instruction-tuned open-weight model can be used as a reward model in this sense; we merely need access to the `compute_logprobs` function.

Distillation as a tool for integrating behaviors and knowledge has also been explored for hybrid reasoning models (Qwen3) and specialist distillation.[37] As our and Chen et al.'s results suggest, on-policy learning can be a crucial tool for augmenting similar distillation-based "model merging" setups.

Discussion

Dense supervision greatly improves compute efficiency

Reinforcement learning and on-policy distillation both learn via reverse KL, pruning the space of actions present in the base policy. The difference is in the density of reward. In LoRA Without Regret we presented the information-theoretic perspective that reinforcement learning only teaches $O(1)$ bits per episode. In contrast, distillation teaches $O(N)$ bits per episode, where N is the number of tokens. Can we quantify the training efficiency gained through denser reward?

We ran an experiment for making a direct comparison between the two:

1. Start with Qwen3-8B-Base (no additional SFT).

2. Run RL on DeepMath, matching our procedure from LoRA Without Regret. We use a LoRA rank of 128. The resultant model is the teacher for distillation.
3. On-policy distill from the RL-trained model (2) back into the base model (1).

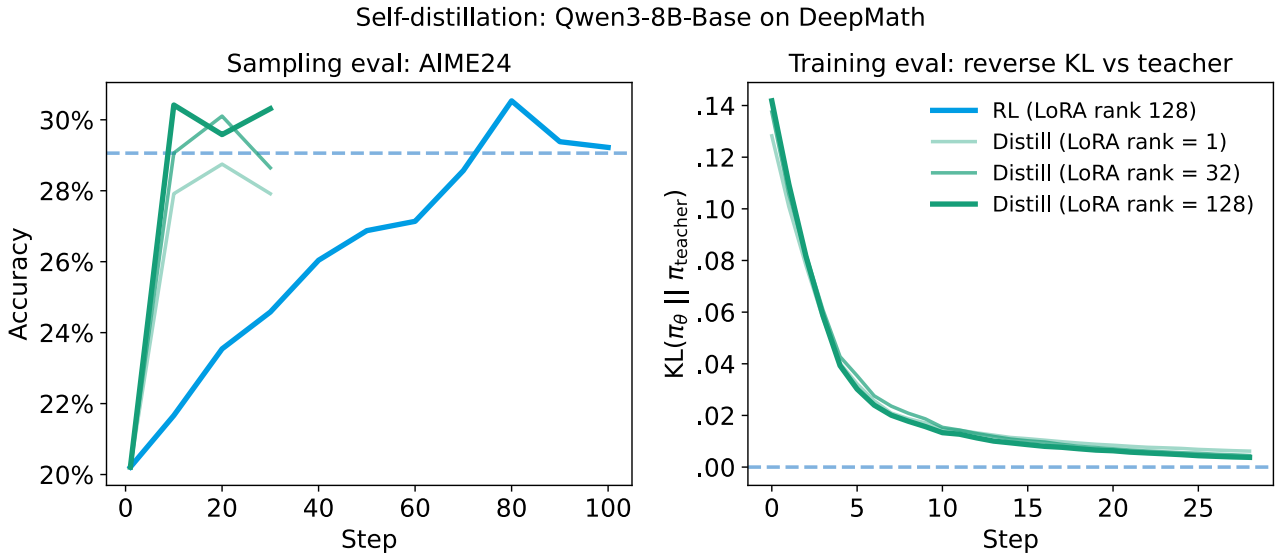


Figure 12: Starting from the same initialization, on-policy distillation can learns the RL-trained policy in approximately 7-10x fewer gradient steps, which corresponds to a compute efficiency of 50-100x.

We see that distillation reaches the teacher's level of performance approximately 7-10x faster than RL with matched model architecture (LoRA rank 128). The reverse KL decreases to near-zero and the AIME score is recovered in under 10 gradient steps, while RL took 70 steps to reach that level.

Cumulatively, the reduction in compute required is on the order of 50-100x:

- While RL requires training at approximately the evaluation context (in order for the policy to learn the context limit and not incur format penalty), distillation learns reasonably at shorter context lengths, as there is no sharp cutoff in reward between a trajectory which has finished sampling and a trajectory that continues.
- When the SFT initialization is strong,[38] on-policy distillation works effectively with much smaller batch sizes since it provides significantly more bits per episode, thereby reducing gradient noise.

Although it is generally difficult to train reinforcement learning models with process supervision, these results indicate that as a broad direction, process supervision and dense rewards have the potential to improve learning efficiency by an order of magnitude. This matches earlier results in RL research from Lightman et al.

Distillation can effectively reuse training data for data efficiency

For practitioners, collecting large datasets of training prompts can be difficult and time-consuming. Therefore, we want to be able to reuse prompts several times in

training. With RL, training multiple epochs on the same prompt often leads to simple memorization of the final answer, especially with large models.[39] In contrast, on-policy distillation learns to approximate the teacher's complete distribution by minimizing reverse KL, rather than memorizing a single answer. This allows us to train many samples from the same prompt.

We repeat the above experiment of training Qwen3-8B-Base on math, but now with only a single randomly chosen prompt from the dataset.[40]

We train on this prompt for 20 consecutive steps, each with a batch of 256 rollouts, for 5120 graded sequences in total. We train on the same prompt for multiple steps in sequential fashion, which normally leads to overfitting. Though this is naturally less compute-efficient, we do approximately match the performance of the teacher model despite only training on a single prompt.

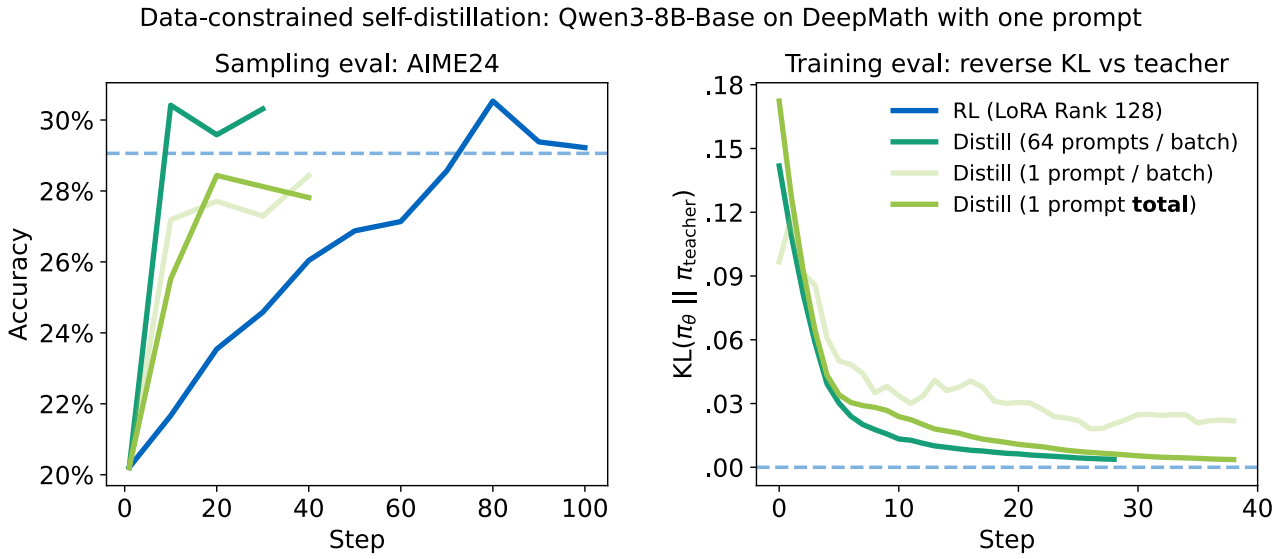


Figure 13: In this example, multi-epoch training on top of one training example is sufficient to distill the teacher's AIME'24 performance. Our default configuration (also used in the personalization experiments) runs on-policy distillation with 64 prompts / batch, and 4 samples / prompt. All methods shown are trained with 256 samples / batch. Note that the right chart is showing training KL, hence it is natural for 1 prompt total to outperform 1 prompt / batch.

RL searches in the space of semantic strategies

We have seen that on-policy distillation can replicate the learning provided by RL with much fewer steps of training. One interpretation of this result is that, unlike pre-training, RL doesn't spend a lot of compute on the gradient steps themselves. We should think of RL as spending most of its compute on *search* — rolling out a policy and assigning credit — rather than on making updates.[41]

Pre-training via stochastic gradient descent is exploring the high-dimensional parameter space. Pre-training requires a vast amount of information and is very

difficult to distill, in part because the parameter space is somewhat unique to each network.[42] The gradient steps required for pretraining are extremely computationally expensive and time-consuming.

In contrast, we should think of RL as exploring the space of semantic strategies.[43] At every step, RL tries a small modification of some strategy it has found in the past. Rather than exploring in the parameter space, it "stumbles" onto new strategies by luck — it is randomly sampling from the set of weights it already has.

Once a good strategy is found, distillation serves as a shortcut for learning it: on-policy distillation does not need to model the intermediate strategies during the curriculum of RL, but rather only the final strategy learned. If we are only interested in the final strategy (common in production use-cases), we need not spend the compute to model all the intermediate ones.

Consider an analogy: in science research, we spend a lot of time and resources looking for answers and exploring new ideas. Once a result is discovered, it is much simpler to teach it to others by expressing it in natural language. We can contrast this to intuitive physical skills such as playing a sport. They are much harder to teach to others, since the knowledge exists in an innate language (e.g., muscle memory) that is only readily understood by ourselves. Sports are only learned with repeated practice.

On-policy learning as a tool for continual learning

In the section on distillation for personalization, we explored the ability of on-policy distillation to re-introduce specialized trained behaviors into the model. This generalizes to a broader set of continual learning tasks, which require acquiring new knowledge without degrading prior capabilities.

Prior work has found on-policy learning (RL) forgets less than off-policy learning.[44] However, RL only shapes behavior — it cannot teach new knowledge well, and thus can't be sufficient for continual learning.

In the above section, we saw that SFT (including off-policy distillation) fails at scaffolding continual learning because it degrades behavior. We investigate this more closely and demonstrate this with a direct example. Similarly to above, we build a dataset by taking Tulu3 prompts and sampling from Qwen3-32B at `temperature = 1.0` and no further modifications. This dataset therefore has a KL of zero against Qwen3-32B.[45]

What happens when we run SFT on this dataset of a model's own samples? We see that any practical learning rate that's greater than zero leads to a degradation of performance on the instruction-following evaluation!

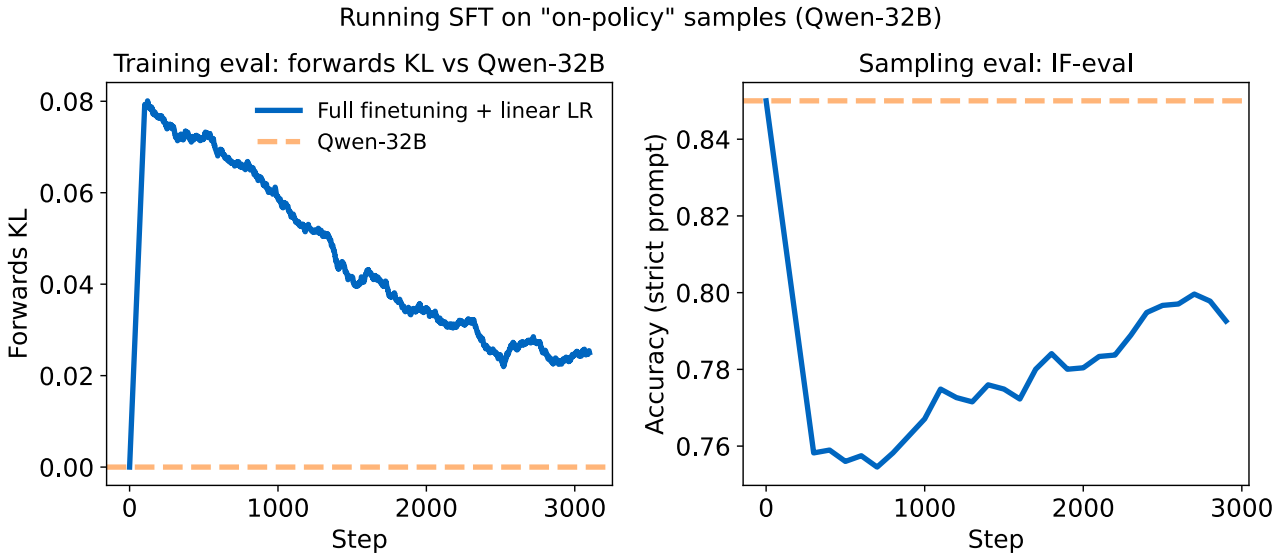


Figure 14: Running SFT on top of Qwen3-32B's own samples degrades performance. We use the same learning rate as from the personalization section, which was swept for practical performance considerations. A linear learning rate can prevent forwards KL / IF-eval from regressing indefinitely, but does not recover performance before the LR decays to zero.

A possible explanation for this is that while KL divergence is 0 in expectation, every finite batch will exhibit a slightly different distribution in practice. Training on these finite batches causes a non-zero gradient update, which then diverges the updated model's policy from that of its original state. This process turns training on one's own samples into off-policy training over time, which causes the same error accumulation and divergence over long sequences that happens with off-policy training.

On-policy distillation always stays on-policy, and since the teacher stays fixed, the student converges on the teacher's desirable behavior, without regressing in the self-distillation setting as SFT does. This makes on-policy distillation a very promising tool for continual learning.

Conclusion

We have explored the application of on-policy distillation for applications such as training a small model for math reasoning or a continuously-learning assistant. We compared on-policy distillation to two other approaches to post-training: off-policy distillation, and on-policy RL. We find that on-policy distillation combines the best of both worlds: the reliable performance of on-policy training, with the cost-efficiency of a dense reward signal.

Post-training is a crucial part of reaching frontier model capabilities. By leveraging on-policy sampling from the student with dense supervision from a teacher, the on-policy distillation recipe reaches those capabilities at a fraction of the cost of frontier high-compute RL runs.

Our implementation can be found in the Tinker cookbook. Our work explored simple and straightforward instantiations of on-policy distillation to clearly showcase its advantages. We hope to continue research on new applications of distillation, new methods for improving teacher supervision, and ways to improve data efficiency and continual learning.

prev back to top next

At Thinking Machines, our mission is to empower people with AI models that combine frontier performance with adaptability and personalization. On-policy distillation is a potent tool for achieving that.

Citation