Data analysis

Ноутбук: Data_analysis.ipynb

Сначала я решил разобраться в gsm8k датасете.

Распределение длины ответов в токенах:

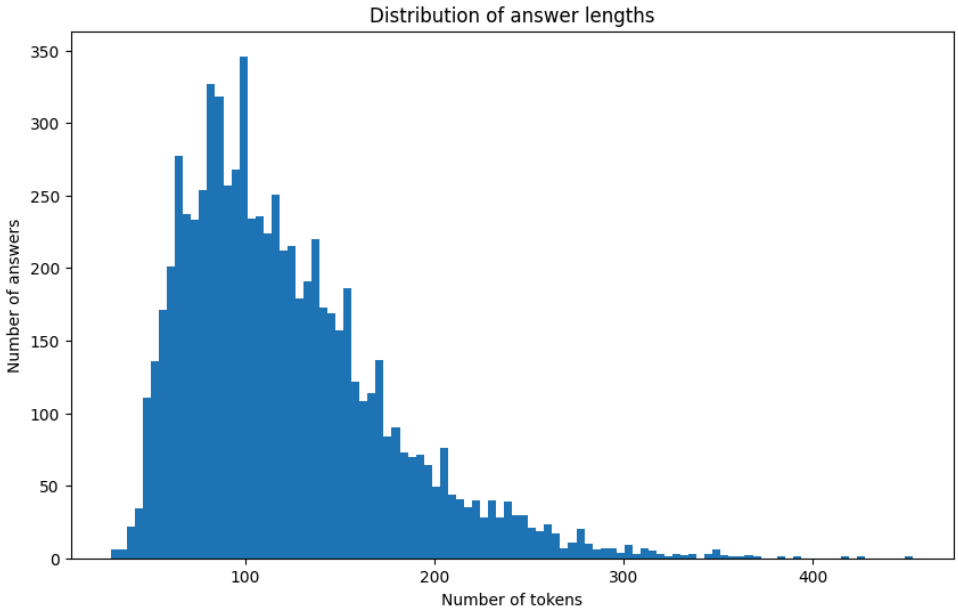


Figure 1: Average length: 121.7933; Minimum length: 29; Maximum length: 453

Распределение длины вопросов в токенах:

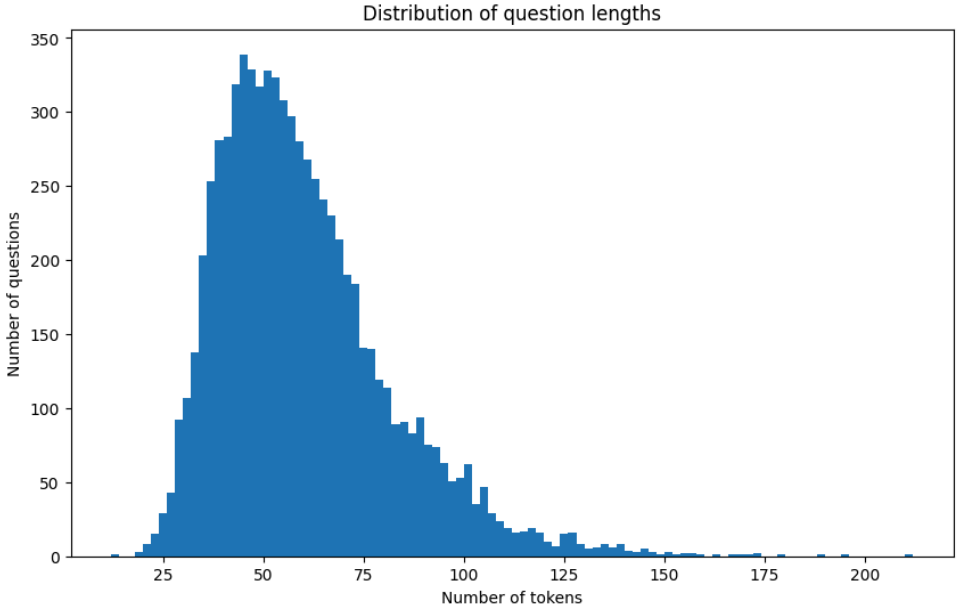


Figure 2: Average length: 59.6473; Minimum length: 12; Maximum length: 212

Эти данные показывают, что ограничение длины ответа модели до 512 токенов является вполне обоснованным. Средняя длина ответа составляет 121 токен, а минимальная — 29 токенов. Это позволяет сделать вывод, что если во время обучения средняя длина ответов начинает снижаться до 30–50 токенов или ниже, то можно говорить о коллапсе политики. В таком случае возможно стоит пересмотреть структуру награды или гиперпараметры обучения. Аналогичная логика работает и в обратную сторону.

В этом датасете также представлена socratic configuration — я решил её использовать, чтобы приблизительно оценить количество логических шагов, необходимых для решения определённой задачи.

Результаты оказались следующими:

- Коэффициент корреляции между длиной вопроса и длиной ответа: 0.5258
- Коэффициент корреляции между количеством требуемых логических шагов и длиной ответа: 0.7711

Это говорит о том, что разработка методов ограничения длины reasoning на основе длины вопроса или числа предполагаемых логических шагов является оправданной. Также можно использовать саму длину ответа в токенах для этого.

Кроме того, я разбил вопросы и ответы на 8 классов в зависимости от количества требуемых логических шагов. Ниже представлена соответствующая статистика (всё в токенах):

Logic Steps	Questions	Mean	Min	Max	Variance	Trimmed Max (97.5%)
2	1960	72.71	29	172	333.36	112
3	2149	105.65	36	221	636.69	163
4	1660	136.06	35	273	1073.77	210
5	954	169.22	51	359	1600.24	258
6	419	196.52	55	369	3021.22	305
7	228	207.52	64	425	5809.29	350
8	90	228.10	89	453	5866.94	366
9	13	240.31	92	367	9781.40	367

Figure 3: Статистика длины ответов в зависимости от количества логических шагов
Trimmed max - это максимальное значение в данных после удаления верхних 2.5% ответов (по длине)

Поскольку я использовал Google Colab, мне удавалось обучать модель максимум на 4000 задачах. Учитывая это ограничение, я решил использовать для обучения только те задачи, в которых число требуемых логических шагов не превышает 5. У нас ограниченное время на обучение, поэтому нецелесообразно тратить его на слишком сложные задачи. Вероятнее всего, на таких задачах модель не сможет выдать ни одного правильного ответа, и, как следствие, во многих реализациях алгоритма GRPO градиенты модели не вовсе обновятся на этой задаче.

Оценивание изначальной модели Qwen2.5-0.5B-Instruct

Ноутбук: Initial_evaluation.ipynb

- Точность: 9.55%
- Количество правильных ответов: 126 из 1319
- Точность формата: 66.34%
- Количество ответов с правильным форматом: 875 из 1319
- Средняя длина ответа: 144.88 токенов

Запуск ноутбуков и загрузка обученных моделей для оценки

Все эти ноутбуки сильно оптимизированы под Google Colab на GPU T4. Их достаточно запустить командой Run all. Они без прерываний проходят этапы загрузки датасетов, обучения, оценки, а в конце скачивают модель и результаты (если всё же возникнет ошибка в оценивание, стоит перезапустить ячейку с ошибкой). Поэтому рекомендуется использовать именно Colab. Не думаю, что стоит создавать ноутбук для локального запуска с requirements.txt, учитывая то, что я сам не смогу проверить работает он или нет и его работоспособность зависит от железа. Чтобы пропустить обучение и сразу запустить оценку модели, надо открыть ноутбук с префиксом Eval в соответствующей папке и запустить через Run All. Сохранить обученные модели для baseline и метода 1 я не успел (colab завершил сессии раньше). Остальные 2 модели сохранил.

Baseline

Ноутбук: Baseline/Baseline_GRPO.ipynb

Изначально я пытался реализовать GRPO, следуя советам из Notion. Однако у меня постоянно возникали проблемы с библиотеками, и мне не удавалось запустить функцию train(). Я предпринимал множество попыток, чтобы заставить код работать — подробно об этом я написал Борису Шапошникову.

Отчаявшись, я нашёл другую реализацию без unsloth, с trl и vLLM в этом видео и решил использовать её. Однако обучение модели в этой реализации занимало значительно больше времени, около 10–15 часов.

Позже я заметил, что даже ноутбуки из официальной документации unsloth не запускаются. На следующий день я решил написать об этом в их дс. Оказалось у них была ошибка в последнем (вроде) пуше и из-за этого библиотека не работала. Они написали что проблема была в пуше от предыдущего дня, но unsloth у меня не работал нормально также 10-11 июля.

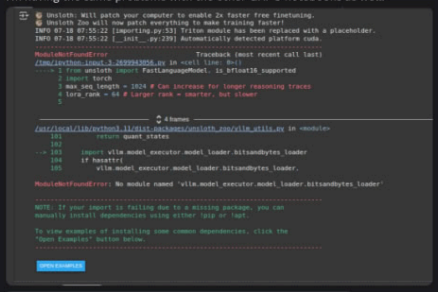
18 July 2025

Sergey 11:01

Hi!

I'm trying to run Qwen2.5_(3B)-GRPO.ipynb from https://github.com/unslothai/notebooks/ on a T4 in Colab using "Run all", but I encountered this error. I've spent several hours trying to fix it, but only ended up with more errors and issues. Could someone please help me with this?

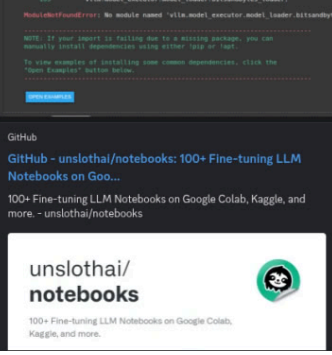
I'm having the same problems with the other GRPO notebooks as well.



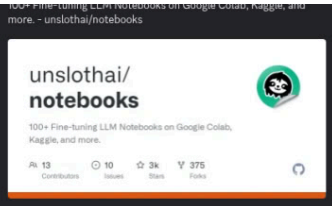
GitHub

GitHub - unslothai/notebooks: 100+ Fine-tuning LLM Notebooks on Goo...

100+ Fine-tuning LLM Notebooks on Google Colab, Kaggle, and more. - unslothai/notebooks



100+ Fine-tuning LLM Notebooks on Google Colab, Kaggle, and more. - unslothai/notebooks



Roland Tannous UnAI 11:17

hello

let me try to reproduce this

Edd UnAI 11:58

@Roland Tannous this is the bug from last night with my push

not sure if it's pushed main yet but we already aware of this error

Roland Tannous UnAI 12:25

oh ok !

thanks for letting me know. i think i was sleeping by then 😆

Daniel UnAI 12:46

ill push pypi asap!

🔥 1

Daniel UnAI 15:50

Done! Please do `pip install --upgrade --force-reinstall --no-cache-dir --no-deps unsloth unsloth_zoo` (edited)

Sergey 18:04

Thank you for your help!

Официальный ноутбук Unsloth не работал с моделью Qwen2.5-0.5B — на каком-то этапе генерации модель полностью прекращала reasoning. Поэтому я решил использовать другие реварды и промпты, в частности из того ноутбука, который не использует unsloth. В результате я значительно улучшил код в обоих ноутбуках: добавил анализ данных, дополнительные метрики, сохранение промежуточных результатов и прочее.

Было бы проще оформить это тестовое задание в виде полноценного Python-проекта, но я решил оставить всё в виде ноутбуков. Причин несколько: моя GPU слишком старая для vLLM, подключение VS Code к Colab, как я понял, затруднительно (если вообще возможно), а запускать тестовое задание для Т-банка на HPC ВШЭ показалось не совсем правильным.

Метрики

Метрики всех методов можно посмотреть в этом wandb отчёте. Чтобы лучше понять разницу в значениях определённых метрик между моделями, рекомендуется использовать сглаживание с помощью скользящего среднего (running average smoothing).

Учитывая опыт работы с ноутбуками, я сначала реализовал всевозможный функционал в бейзлайн ноутбуке, а затем немного модифицировал его под каждый из методов. Хотя GRPOTrainer логирует множество метрик в wandb, я решил написать собственный класс метрик — my_metrics. В нём, помимо стандартной accuracy, реализованы:

1. question_vs_response_length_correlation — метрика, отслеживающая корреляцию между длиной (в токенах) вопроса и длиной ответа LLM.
2. answer_vs_response_length_correlation — корреляция между длиной (в токенах) правильного ответа из датасета и длиной ответа, сгенерированного LLM.
3. out_of_limit — доля генераций, превысивших максимальную длину или максимально близких к лимиту (меньше 3 токенов)
4. model_efficiency $= \frac{\text{answer\_sum}}{\text{response\_sum}}$, где
   - answer_sum — суммарная длина (в токенах) правильных ответов из датасета, на которые LLM также ответила верно,
   - response_sum — это суммарная длина (в токенах) всех тех ответов, которые были сгенерированы моделью (LLM) и при этом оказались правильными.

   Таким образом, чем выше model_efficiency, тем эффективнее работает модель.

Во время обучения я в онлайне выводил результаты работы LLM на каждом батче. (кроме бейзлайн ноутбука)
Обозначения:
✅ — модель дала правильный ответ на вопрос
⭐ — ответ имеет правильный формат
❌ — либо неверный ответ на вопрос, либо неправильный формат

Гиперпараметры
Вычислительных ресурсов было мало, поэтому гиперпараметры я почти не подбирал вручную, а использовал те, что были в ноутбуке из видео. В целом, причины выбора этих гиперпараметров понятны.

Отдельно отмечу возможную причину выбора значения beta = 0.005 в GRPOConfig. Часто beta ставят равной 0 — то есть полностью исключают KL-дивергенцию. Однако в данном случае, вероятно из-за других гиперпараметров (lr) и нестабильной реварда функции, при beta = 0 текущая политика модели слишком сильно отклонялась от референтной, и из-за этого чаще происходило clipping, что ухудшало обучение. Поэтому небольшое значение beta = 0.005, скорее всего, выбрано для того, чтобы сохранить контроль над отклонением политики и уменьшить частоту clipping.

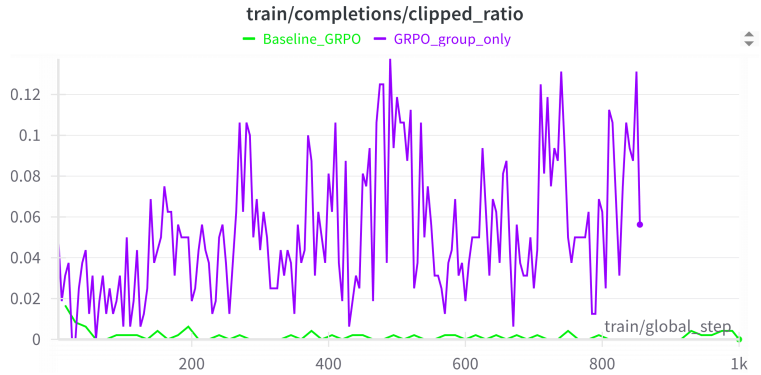


Figure 4: GRPO_group_only - один из первых запусков второго метода (о нём дальше), с большим ревардом и относительно маленькой beta.
Baseline_GRPO - запуск бейзлайн-ноутбука с подходящим значением β

Я выбрал ‘constant_with_warmup’ в качестве learning rate scheduler, чтобы модель могла полноценно обучаться почти на всех 4000 вопросах.

Сид рандома фиксирован, LLM всегда получают одинаковые входные данные в одинаковом порядке. (Сид не подбирался)

Результаты модели:
- Точность: 34.50%
- Количество правильных ответов: 455 из 1319
- Точность формата: 98.94%
- Количество ответов с правильным форматом: 1305 из 1319
- Средняя длина ответа: 124.06 токенов
- Время обучения: 11,000 секунд (3 часа 3 минуты)

Таким образом, базовое решение увеличивает точность ответов LLM примерно в 3.5 раза, а точность формата — в 1.5 раза.

В целом, я уверен, что более точным подбором гиперпараметров модели (в частности значения награды за правильный ответ) можно значительно улучшить точность (до 40-45%). Но у нас немного другая задача.

Анализ возможных решений

В целом задача сводится к уменьшению длины reasoning при сохранении или улучшении точности (accuracy) ответов LLM. Это непросто: с одной стороны, как показано в работе [1], более длинные рассуждения часто коррелируют с более высокой точностью. С другой стороны, как показывает [2], такая корреляция не всегда держится, и в ряде случаев более короткие рассуждения дают более точные ответы.

Жёсткое ограничение длины инференса

Есть несколько простых идей, как можно ограничивать длину ответа модели перед инференсом:

1) Используя тренировочный датасет, каждому вопросу сопоставить количество логических шагов необходимых для его решения. Затем при инференсе разрешать модели использовать столько токенов, сколько было максимум (или trimmed max) у ответов с таким же числом шагов.

2) Ограничить длину ответа длиной правильного ответа из датасета на этот же вопрос.

3) Оценить нужную длину ответа по длине вопроса. Например, разбить вопросы на группы по длине и задать ограничение по длине ответа для каждой группы используя тренировочный датасет.

Дополнительно можно прямо в prompt написать: “Use no more than N tokens to answer the question”,

Минусы

(эти минусы могут отсутствовать в специфичных реализациях GRPO)

- Ограничение на длину инференса — очень жёсткое. Если модель выходит за установленный предел, она зачастую получает reward = 0 (во многих реализациях GRPO). Даже если формат ответа модели полностью соответствует требованиям, она всё равно не получит полный reward за формат.
- Кроме того, такая схема не различает два случая: если модель превысила лимит всего на 10 токенов, или на 100 — оба варианта наказываются одинаково, хотя интуитивно второе нарушение должно быть санкционировано сильнее.
- Такие ограничения уменьшают пространство допустимых политик поведения модели. Возможно, что при более длинных рассуждениях точность (accuracy) существенно возрастает. Также в ходе обучения может оказаться полезным сначала позволять модели делать длинные рассуждения, а затем постепенно учить её сокращать их. Такие стратегии становятся невозможными при жёстких ограничениях на длину инференса.

Плюсы

- Простота реализации

Ограничение длины через reward

Мне показалось, что у предыдущего метода есть серьёзные недостатки, которые можно обойти, регулируя длину ответа через reward.

Плюсы

- Отсутствуют проблемы с reward за соблюдение формата.
- Мы можем по-разному наказывать модель в зависимости от длины её ответа.

- Пространство возможных политик расширяется: если LLM «считает», что использование более длинного рассуждения заметно повышает точность (accuracy) на текущем этапе обучения и способно компенсировать штраф за длину, она может свободно выбрать такой вариант.
- Можно использовать данные из других генераций в группе в рамках GRPO.

Минусы

- Реализация становится сложнее — надо сделать reward за длину, который не приведёт к коллапсу политики или reward hacking.
- Потенциально увеличивается дисперсия reward.

Учитывая эти соображения, а также пункт 2 из правил, я решил сосредоточиться на подходе с ограничением длины через reward. На этот выбор также повлиял тот факт, что, когда я впервые читал [1], мне очень понравилась идея оценивать value-функцию на основе разных ответов на один и тот же вопрос. Мне показалось, что такие ответы можно использовать не только для этой цели и хотелось углубиться в эту тему.

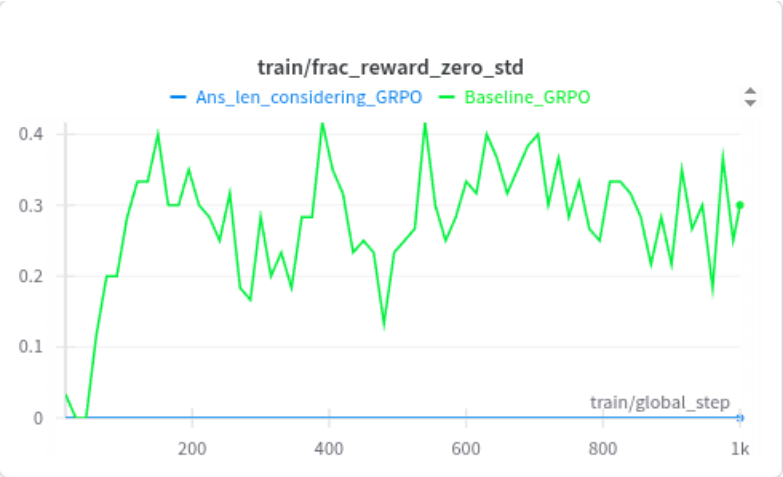
1 метод

Ноутбук: Ans_len_considering/Ans_len_considering_GRPO.ipynb

Самое простое, что можно реализовать — это штраф за отклонение длины сгенерированного ответа LLM от длины правильного ответа из тренировочного датасета:
$r_i = -\frac{|o_i^{\text{gen}} - o_i^{\text{gt}}|}{\max_j |o_j^{\text{gen}} - o_j^{\text{gt}}|} \cdot \lambda$
где:
o_i^{gen} - длина сгенерированного ответа
o_i^{gt} - длина правильного ответа из тренировочного датасета

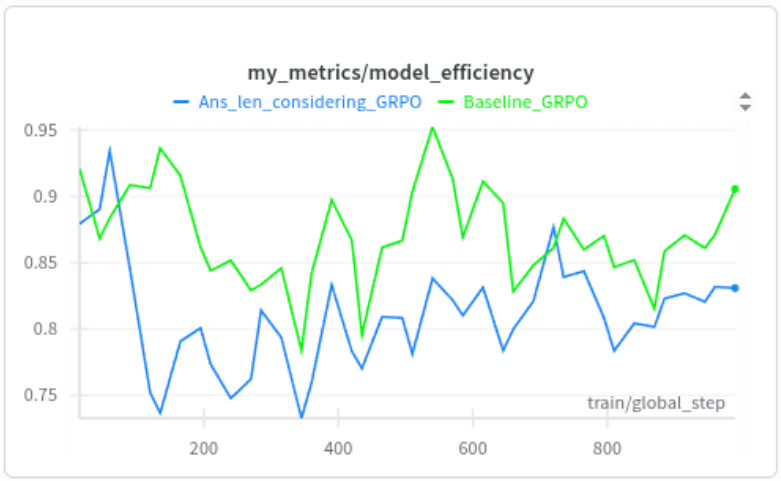
Плюсы

- Предотвращает коллапс политики.
- Стимулирует рассуждения там, где они действительно нужны, и наказывает там, где они излишни.
- У unsloth есть метрика frac_reward_zero_std — это доля групп сгенерированных ответов, у которых стандартное отклонение вознаграждений равно нулю. В таких группах, скорее всего, градиенты обновления будут равны нулю, поскольку reward и value function совпадают. При baseline GRPO эта метрика принимает достаточно большие значения, тогда как при методе 1 её значение очевидно всегда равно нулю. Это означает, что в baseline мы не используем для обучения примерно 30% данных, а при методе 1 все данные используются хоть в какой-то степени.



Минусы

- Даже в группах, разбитых по количеству логических шагов, дисперсия длины ответов очень высокая. Поэтому такой метод может запутать модель и привести к высокой дисперсии reward.
- Модель может придумать более короткие паттерны рассуждений и за них получать отрицательный reward.
- Используется длина правильного ответа, но не во всех датасетах дано полное правильное решение
- Видимо из-за большой дисперсии эффективность модели тоже падает сильно



Результаты модели

Точность: 34.34%
Правильных ответов: 453 из 1319
Точность формата: 99.62%
Правильных по формату: 1314 из 1319
Время обучения: 10540 секунд (2 часа 55 минут)

Точность модели практически не изменилась. По времени обучения и графику эффективности видно, что модель не стала заметно эффективнее или медленнее. Понятно, что за счёт различных стратегий разбиения на подгруппы можно уменьшить дисперсию награды и, возможно, улучшить эффективность метода.

2 метод

Ноутбук: Group_method/Group_method_GRPO.ipynb

В GRPO при оценке reward нам даётся группа размером batch_size × num_generations. Каждую подгруппу для отдельного вопроса размером num_generations будем рассматривать отдельно.

Пусть:

- o_i - длина i-й генерации в подгруппе
- $o_{\min}$ - минимальная длина ответа, среди всех правильных ответов LLM в подгруппе
- correct - среднее значение корректности
- λ - коэффициент штрафа за длину

Тогда:

1) $r_i = -\lambda \text{ correct } \frac{\max(0, o_i - o_{\min})}{\max_j(\max(0, o_j - o_{\min}))}$

2) Если $\max_j\big(\max\big(0, o_j - o_{\min}\big)\big) = 0$, то reward за длину для всех элементов подгруппы = 0

Объяснение:

Когда разрабатываешь формулу для reward, задаёшься вопросом: почему именно такая формула должна улучшить процесс обучения LLM? Что нового она даёт модели, чего она не может выучить в процессе обычного alignment?

Представим, что LLM сгенерировала ответы длиной 30, 40, 60 и 100. Из них только ответ длиной 40 оказался правильным. В стандартном GRPO мы накажем остальные ответы (30, 60, 100) одинаково. Тем самым мы не передадим модели дополнительную информацию о том, что правильный ответ можно сгенерировать, используя 40 токенов.

Но если мы знаем, что существует корректный ответ длиной 40, то зачем LLM генерировать ответ длиной 100? Избыточный reasoning может только запутать модель.

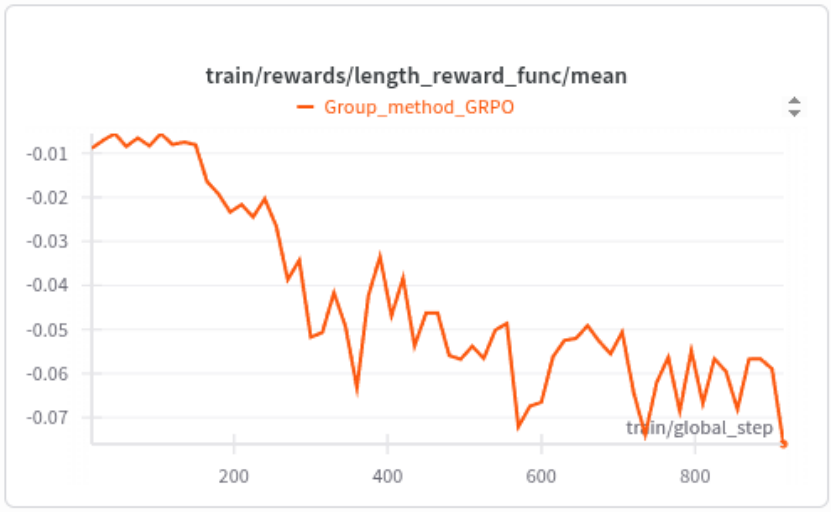
При этом наша цель повысить эффективность инференса, поэтому мы штрафуем только за избыточную длину reasoning. В дополнение к этому, если LLM из num_generations попыток ответила правильно только один раз, мы не хотим сильно стимулировать её сокращать длину reasoning для этого вопроса, так как пока модель плохо отвечает на этот вопрос в целом. И наоборот: если большинство генераций оказались правильными, мы можем позволить себе более агрессивное сокращение длины. Именно поэтому мы домножаем штраф на значение correct.

Плюсы

- Передаёт в LMM дополнительную информацию о вопросе через генерации в подгруппе
- Не требует полного правильного ответа для работы
- model_efficiency стабильно выше, чем у бейзлайна (см. Figure 6)
- frac_reward_zero_std в середине обучения становится заметно меньше, чем у бейзлайна (скорее всего, потому что теперь, если LLM ответила все num_generations раз правильно на вопрос, то мы обновим веса модели, в отличие от бейзлайн)
- Во время обучения с помощью метода 2 LLM сгенерировала на 30000 меньше токенов (см. Figure 6)
- При этом accuracy остаётся на том же уровне

Figure 7: К сожалению, вычислительных ресурсов Colab оказалось недостаточно, чтобы завершить обучение. Пришлось остановить обработку после примерно 3700 вопросов.

- Мы можем изменять дисперсию и робастность награды, заменяя correct на correct^p и $o_{\min}$ на другие функционалы, например, на медиану, среднее или α-кватиль.
- В начале обучения штраф за длину очень мальнький (из-за умножения на correct), поэтому в начале обучения наш метод не мешает LLM обучаться.



Минусы

- Повышение дисперсии reward (использование correct и $o_{\min}$)
- Вероятность коллапса политики увеличивается (по метрикам видно, что это фактически происходит в самом начале обучения (не повезло с сидом рандома на самом деле), но затем модель восстанавливается)

Результаты модели

- Точность: 34.95%
- Количество правильных ответов: 461 из 1319
- Точность формата: 99.32%
- Количество ответов с правильным форматом: 1310 из 1319
- Средняя длина ответа: 121.6 токенов
- Время обучения: 9816 секунд (2 часа 43 минуты)

Таким образом, мы наблюдаем сокращение средней длины ответа и уменьшение времени обучения на 20 минут (экономим 11% от времени обучения бейзлайн модели). При этом accuracy даже становится немного больше.

Я бы очень хотел протестировать этот reward при разных сидах рандома, при улучшенных гиперпараметрах базовой модели, при LLM на 3B параметров. Но, к сожалению, в Colab это всё занимает слишком много времени и сил и я не успеваю.

1 + 2 метод

Ноутбук: Group_method_with_ans_len_considering/
Group_method_with_ans_len_considering_GRPO.ipynb

Плюсы

- Сочетание двух методов может снизить вероятность коллапса политики, возникающего из-за второго метода, и уменьшить потерю эффективности, связанную с первым методом.

Минусы

- При сложении двух методов reward суммируются и их дисперсии (если считать, что они независимы).

Результаты модели

Точность: **35.41%**
Количество правильных ответов: 467 из 1319
Точность формата: 99.85%
Количество ответов с правильным форматом: 1317 из 1319
Средняя длина ответа: 122.80 токенов
Время обучения: 10240 секунд (2 часа 50 минут)

Таким образом, сочетание двух методов позволяет достичь наивысшей accuracy. При этом мы сокращаем время обучения по сравнению с бейзлайн-моделью на 7%.

Заключение

Мы показали, что при обучении модели Qwen2.5-0.5-Instruct на датасете openai/gsm8k применение метода 2, а также комбинации методов 1 и 2, увеличивает скорость генерации ответов, ускоряет процесс alignment в целом и при этом сохраняет accuracy.

Я также провёл небольшой анализ изменений в структуре ответов на тестовые вопросы, но, кажется, и без того получилось слишком объёмно.

Очень краткий анализ литературы

Обычно я начинаю работу над новой задачей с чтения статей, но в этот раз решил сначала самостоятельно исследовать возможные методы решения задачи и только после запуска методов 1 и 2 я начал искать статьи.

Я не нашёл большого количества статей про оптимизацию эффективности reasoning через reward в GRPO. Скорее всего, это связано с тем, что GRPO относительно новый алгоритм ([3]), а задание различных ревардов и их масштабирование на действительно большие языковые модели — задача очень нетривиальная. Ближе всего к моему реварду — ревард из [4], но там не учитывается отрицательный ревард за неправильные ответы, что крайне важно.

generated responses $\{o_i\}$, we first isolate the subset of correct responses and compute the mean μ and standard deviation σ of their token lengths. For a correct response o with length $|o|$, we define its standardized length deviation as:

$$z = \frac{|o| - \mu}{\sigma + \epsilon}, \tag{2}$$

where $\epsilon > 0$ is a small constant added for numerical stability. The final reward is modulated using an exponential decay function:

$$R_{\text{accuracy}}(o|q) = \begin{cases} \exp(-\alpha z), & \text{if } o \text{ is correct,} \\ 0, & \text{if } o \text{ is incorrect.} \end{cases} \tag{3}$$

where $\alpha > 0$ is a tunable hyperparameter controlling the strength of length penalization.

Figure 9: Reward from [4]

$$\text{len\_reward(i)} = \begin{cases} \lambda & \text{If } r(x, y_i, y^*) = 1 \\ \min(0, \lambda) & \text{If } r(x, y_i, y^*) = 0 \end{cases}, \quad \text{where } \lambda = 0.5 - \frac{\text{len}(i) - \text{min\_len}}{\text{max\_len} - \text{min\_len}}.$$

Figure 10: Reward from [5]

Также в начале я ссылался на [2]. В этой статье показано, что большая длина reasoning на математических задачах не всегда приводит к более высокой accuracy. В частности, это хорошо демонстрирует Figure 10.

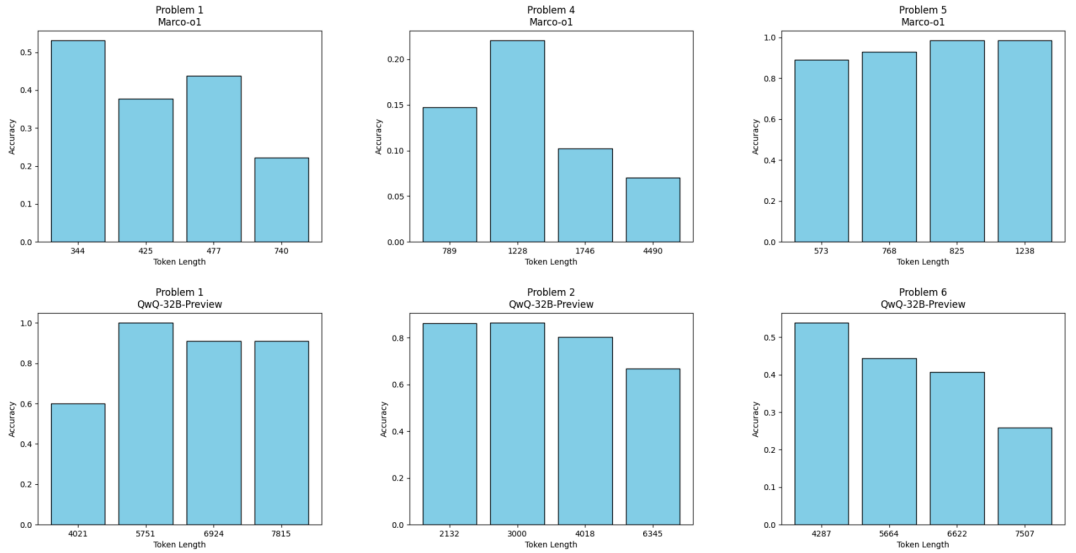


Figure 11: Accuracy-Length Relationship at Instance level from [2]

Bibliography

[1] DeepSeek-AI *et al.*, "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning," 2025. https://arxiv.org/abs/2501.12948.

[2] H. Luo *et al.*, "O1-Pruner: Length-Harmonizing Fine-Tuning for O1-Like Reasoning Pruning," 2025. https://arxiv.org/abs/2501.12570.

[3] Z. Shao *et al.*, "DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models," 2024. https://arxiv.org/abs/2402.03300.

[4] J. Zhang and C. Zuo, "GRPO-LEAD: A Difficulty-Aware Reinforcement Learning Approach for Concise Mathematical Reasoning in Language Models," 2025. https://arxiv.org/abs/2504.09696.

[5] K. Team *et al.*, "Kimi k1.5: Scaling Reinforcement Learning with LLMs," 2025. https://arxiv.org/abs/2501.12599.