

Выбор задачи + личная мотивация

Для тренировки LLM агента с использованием RL я решил выбрать задание №2 ЕГЭ по информатике на восстановление таблицы истинности по заданной логической функции и фрагменту данных подходит для наших условий для среды. Ответ на любое задание верифицируемый простой проверкой. Мы можем регулировать сложность разным количеством переменных, сложностью выражения, количеством данных в таблице и т.д. и среда решается за один запрос в LLM. К тому же, сейчас я в процессе подготовки к ЕГЭ по информатике, и мне самому интересно узнать насколько хорошо такой подход сможет противостоять среднестатистическому школьнику.

Архитектура проекта и назначение модулей

Уровень абстракции base/ - общие правила, по которым работают все части проекта

- base/env.py - базовый класс среды, определенный стандарт который говорит, что любая среда должна уметь генерировать задачу и проверять ответ
- base/verifier.py - интерфейс системы проверки ответов
- base/data.py - контейнер для данных, упаковывает вопрос, ответ, и метаданные в единый объект для дальнейшего использования

Реализация среды envs/ - тут сосредоточена вся логика, относящаяся к заданию №2 ЕГЭ.

- envs/ege_logic_env.py - основной файл, он вызывает генератор формул, строит truth table, составляет итоговое задание
- envs/formula_generator.py - собирает формулы из логических операций (and, or, xor... и т.д.)
- envs/truth_table.py - модуль который перебирает 0 и 1 для всех переменных и строит на основе этого таблицу истинности
- envs/fragment_builder.py - модуль который следит за тем, чтобы в фрагменте таблицы было достаточно данных для решения задачи, если столбцы нельзя различить, то модуль бракует задачу
- envs/[difficulty.py](#) - конфигуратор сложности, отвечает за то сколько переменных и строк таблицы давать на каждом из уровней 1-10
- envs/prompt_builder.py - из формулы и чисел в таблице истинности создает красивый промпт для LLM агента, который он поймет

Проверка ответов verifiers/ - совпадает ли ответ нейронки с правильным решением

- verifiers/ege_logic_verifier.py - модуль который читает ответы нейросети и находит сопоставление переменных и сравнивает это с эталонным решением

Модуль scripts/ - подручные скрипты и утилиты которые я использовал

- scripts/build_test_sets.py - скрипт который генерирует задачи разных сложностей и сохраняет их в датасеты
- scripts/test_generate.py - быстрая проверка того что генерирует ege_logic_env.py

Модуль datasets/ - папка с jsonl датасетами

- datasets/test_easy.jsonl - папка с легкими задачами

- datasets/test_medium.jsonl - папка со средними задачами
- datasets/test_easy.jsonl - папка со сложными задачами

Для генерации задач мы выбираем уровень сложности (1-10). На основе этого среда определяет сколько переменных будет в задаче (от 2 до 5), сложность логической формулы и количество строк которое увидит модель. После этого среда рекурсивно строит логическое выражение. Используется дерево случайных операций, программа выбирает случайную связку операций и наращивает на нее левую и правую части. Следующий шаг - убедиться, что у нас реально чистая формула, то есть результат формулы меняется при изменении каждой из переменных, если же у нас какая-то переменная не влияет на результат формулы, то такая формула выбрасывается, и новая формула генерируется.

После этого мы должны создать саму задачу, для этого из полной таблицы истинности выбирается случайный набор строк и перемешивается, после этого программа понимает, можно ли однозначно сопоставить переменные, в обратном случае программа добавляет по одной новой строчке пока нельзя будет их сопоставить. После этого формула и фрагмент таблицы истинности интегрируются в промпт и у нас получается задача по типу задания №2 ЕГЭ по Информатике.

Алгоритмическая верификация среды Solver.

Перед тем как переходить к обучению модели, нам нужно повторно убедиться, что все работает математически без ошибок. Для этого был создан класс Solver, который использует метод брутфорса. Он берет формулу и фрагмент таблицы из метаданных задачи и перебирает всевозможные комбинации переменных для того чтобы убедиться, что решение действительно можно найти по фрагменту таблицы и оно является уникальным.

Каждая задача из трех сетов (Easy, Medium, Hard) прошла проверку, и это показало, что каждую задачу действительно можно решить по данному фрагменту таблицы истинности и решение для каждой задачи единственное.

Baseline evaluation

В качестве baseline-а я взял модель Qwen2.5-1.5B-Instruct чтобы проверить как оригинальная модель без какого-либо дообучения справится с задачей.

Результаты Baseline:

- Easy - 43.33%
- Medium - 13.33%
- Hard - 10%

Обучение с подкреплением (RL)

Для дообучения модели под нашу задачу я использовал алгоритм GRPO (Group Relative Policy Optimization). Основная цель - заставить модель генерировать несколько вариантов решения одновременно и поощрять те цепочки мыслей, которые приводят к верному ответу.

Методика:

Железо - я обучал модель локально, потому что предполагал, что обучение может занять довольно долгое время, а из-за возможных перебоев сети и перегрузки ОЗУ (мы все-таки работаем LLMs), прогресс может остановиться, поэтому я решил запустить обучение локально.

Награда (Process-based reward)

- Награда за формат: 0.2 балла за правильное использование тегов <think> и <answer>
- Награда за правильный ответ: 1 балл за правильное сопоставление переменных

Основная идея использования process-based reward - это дополнить обычный outcome reward. Мы хотим попробовать заставить модель думать и структурированно подходить к задаче. Основная цель RL для обучения Large Language Models - менять policy для того чтобы генерировались токены которые будут приводить к высокому reward с большей вероятностью. Гипотеза заключается в том, что если мы поощряем thinking модели, то она будет пытаться строить какие-то логические связи и последовательно стремиться прийти к правильному ответу.

Как проходило обучение?

Этап обучения, шаги	Средняя длина токенов	Format reward (0-0.2)	Correctness reward (0-1.0)	Интерпретация
0-50	40-120	0.04	0.05	Поиск формата, короткие случайные ответы
50-150	250-450	0.12	0.15	Попытка рассуждений, длинные ответы, логические рассуждения
150-250	150-300	0.20	0.35 (бывают пики 0.5+).	Оптимизация, более структурированные и постоянные ответы

Анализ:

- Длина генерации текстов: на начальном этапе модель генерировала быстрые ответы и игнорировала теги <think>. К середине обучения наблюдался пик длины токенов потому что модель начала пробовать разные стратегии рассуждения. К финалу длина стабилизировалась, так как модель научилась находить решения более эффективно и меньше “воды”
- Модель быстро научилась следовать формату и поняла, что ее поощряют за соблюдение правильной структуры
- До 100 шага награда за правильный ответ была нестабильной, преобладал “шум”, но после этого количество правильных ответов увеличилось. Несмотря на то, что baseline работает лучше, можно увидеть, что у модели действительно были улучшения в том как она решает эту задачу.

После завершения 250 шагов, у модели получились следующие результаты:

Сложность вопроса	Baseline Qwen	Qwen после GRPO
Easy	43.33%	36.67%
Medium	13.33%	13.33%
Hard	10%	3.33%

Анализ цифр:

Дисциплина формата - Модель хорошо научилась работать с форматом thinking-a, она стабильно начала использовать теги <answer> и <think> и получать награду за них соответственно

Появление размышлений - В ответах модели появились реальные попытки логического вывода, в которых не было baseline, в процессе обучения действительно была тенденция на то, что модель учится думать, длина ее ответов выросла с 50 токенов до 350+.

Получается действительно она начала больше думать и перебирать разные исходы в тегах <think>

Trade-off: Модель в процессе обучения потеряла часть “интуитивной” точности от базовых настроек, но не натренировалась достаточно отлаженно решать логическую систему.

Гипотеза - это то, что 250 шагов не достаточно чтобы полностью поменять логику работы модели.

Пример:

Задача:

You are a logical reasoning assistant.

Your task is to identify which variable corresponds to which column in a truth table.

Boolean function:

$$F = (((\text{not } x) \text{ or } (\text{not } y)) \wedge ((\text{not } y) \text{ or } z)) \wedge (\text{not } z))$$

Fragment of the truth table (columns are permuted):

z | y | x | F

1 | 0 | 0 | 1
0 | 1 | 1 | 0
0 | 1 | 0 | 0
1 | 1 | 1 | 1
1 | 0 | 1 | 0
1 | 1 | 0 | 1
0 | 0 | 0 | 1

Variables to assign: x, y, z

Columns labels: z, y, x

Instructions:

- 1. Analyze the Boolean function for all possible inputs.*
- 2. Match the columns in the fragment to the variables x, y, z...*
- 3. Provide your final answer in the format: x=column_name, y=column_name...*

Final answer MUST be wrapped in <answer> tags.

Example: <answer>x=A, y=B, z=C</answer>

Оригинальный Qwen2.5-1.5B	Дообученный Qwen2.5-1.5B
<p>The given Boolean function and truth table provide us with specific values for each combination of x, y, and z. We need to match these values to the corresponding columns labeled as z, y, and x.</p> <p>Let's analyze the function step by step:</p> <p>1. For input (x=0, y=0, z=0): The output should be 1.</p> <ul style="list-style-type: none">- Not(x) = 1- Not(y) = 1- Not(z) = 1- (Not(x) or Not(y)) = 0- (Not(y) or z) = 0- (Not(z)) = 0	<p>The given Boolean function and truth table provide us with information about how different combinations of x, y, and z affect the output F. We need to match each input combination from the truth table to its corresponding variable.</p> <p>Let's analyze the function step by step:</p> <ul style="list-style-type: none">- For z=1, y=0, x=0: The expression evaluates to 1.- For z=0, y=1, x=1: The expression evaluates to 0.- For z=0, y=1, x=0: The expression evaluates to 0.- For z=1, y=1, x=1: The expression

<p>- $((\text{Not}(x) \text{ or } \text{Not}(y)) \wedge ((\text{Not}(y) \text{ or } z))) \wedge (\text{Not}(z)) = 0 \wedge 0 \wedge 0 = 0$ This does not match the expected value of 1.</p> <p>2. For input (x=0, y=0, z=1): The output should be 0.</p> <ul style="list-style-type: none"> - $\text{Not}(x) = 1$ - $\text{Not}(y) = 1$ - $\text{Not}(z) = 0$ - $(\text{Not}(x) \text{ or } \text{Not}(y)) = 0$ - $(\text{Not}(y) \text{ or } z) = 1$ - $(\text{Not}(z)) = 0$ - $((\text{Not}(x) \text{ or } \text{Not}(y)) \wedge ((\text{Not}(y) \text{ or } z))) \wedge (\text{Not}(z)) = 0 \wedge 1 \wedge 0 = 0$ This matches the expected value of 0. <p>3. For input (x=0, y=1, z=0): The output should be 0.</p> <ul style="list-style-type: none"> - $\text{Not}(x) = 1$ - $\text{Not}(y) = 0$ - $\text{Not}(z) = 1$ - $(\text{Not}(x) \text{ or } \text{Not}(y)) = 1$ - $(\text{Not}(y) \text{ or } z) = 1$ - $(\text{Not}(z)) = 0$ - $((\text{Not}(x) \text{ or } \text{Not}(y)) \wedge ((\text{Not}(y) \text{ or } z))) \wedge (\text{Not}(z)) = 1 \wedge 1 \wedge 0 = 1$ This does not match the expected value of 0. <p>4. For input (x=0, y=1, z=1): The output should be 1.</p> <ul style="list-style-type: none"> - $\text{Not}(x) = 1$ - $\text{Not}(y) = 0$ - $\text{Not}(z) = 0$ - 	<p>evaluates to 1.</p> <ul style="list-style-type: none"> - For z=1, y=0, x=1: The expression evaluates to 0. - For z=1, y=1, x=0: The expression evaluates to 1. - For z=0, y=0, x=0: The expression evaluates to 1. <p>Now let's match these results to the columns in the truth table:</p> <ul style="list-style-type: none"> - When z=1, we get F=1, so this must correspond to row where z=1. - When y=0, we get F=1, so this must correspond to row where y=0. - When x=0, we get F=0, so this must correspond to row where x=0. - When z=0, we get F=0, so this must correspond to row where z=0. - When y=1, we get F=0, so this must correspond to row where y=1. - When x=1, we get F=1, so this must correspond to row where x=1. - When z=0, we get F=1, so this must correspond to row where z=0. <p>From this analysis, we can see that:</p> <ul style="list-style-type: none"> - z corresponds to the first column - y corresponds to the second column - x corresponds to the third column <p>Therefore, the final answer is: <answer>x=z, y=y, z=x</answer></p>
--	---

Анализ примера:

Сравнение двух моделей показывает, что Baseline модель страдает от неэффективного использования токенов. Из-за неэффективного перебора, модель достигает лимита в 512 токенов до того как приходит к ответу.

В то же время дообученная модель фокусируется только на важных условиях, это позволяет ей быть в установленных лимитах на токены. GRPO не просто учит модель

запоминать правильные буквы, он оптимизирует мыслительный процесс модели таким образом чтобы она хорошо подходила под условия задачи.

Заключение

У меня получилось научиться автоматически генерировать задачи по типу Задачи №2 ЕГЭ. Также еще у меня первый раз получилось запустить LLM локально и запустить ее дообучение. Даже у небольшой модели Qwen2.5 1.5B получилось начать дообучение и она действительно начала строить какую-то логику.

Что можно сделать дальше?

- Мы можем взять модель на большее количество гиперпараметров, потому что мощностей Qwen2.5 1.5B может просто нехватать на то чтобы держать во внимании сложные цепочки рассуждений. Использование моделей уровня Qwen2.5-7B или Llama-3-8B на более мощных GPU (Я делал запуск на своем личном ноутбуке локально, обучение заняло 8.5 часов)
- Увеличение шагов: 250 шагов хоть они работали на моем девайсе локально 8.5 часов - это все равно очень мало, модель к этому моменту только разобралась с форматом, логично было бы продолжить обучение дальше и посмотреть как будет изменяться награда и поведение модели.
- Усложнение системы наград: сейчас модель получает либо 1 либо 0 за правильный ответ, можно попробовать награждать ее частичными баллами за правильное сопоставление только части переменных. Там тоже есть пространство для воображения как менять функцию сколько модель будет получать (линейно, экспоненциально)
- Можно поэкспериментировать с наградой за формат, то есть например добавить награду за теги <truthtable> и <matchingvariables> чтобы попробовать заставить модель стремиться перебирать различные комбинации переменных, подобрать и тестировать свои гипотезы
- Можно запускать обучение модели поэтапно, вместо того чтобы давать модели задачи рандомной сложности, можно ее поэтапно увеличивать. То есть например сначала добиться того чтобы модель хорошо научилась решать задачи сложности 1-2, и потом постепенно ее увеличивать
- Есть потенциал расширить среду на другие задачи егэ и обучить модель прорешивать ЕГЭ по информатике, чтобы реально у нас получилось сделать модель которая будет не так много весить, но при этом будет справляться лучше или на уровне с определенными задачами по сравнению с огромными LLM моделями. Тут мне кажется действительно заложен потенциал обучения LLM агентов которые будут обучаться не на огромных датасетах с текстами, а на средах. Они будут меньше, быстрее и лучше
- Дополнительно еще я изначально задумывался над этим экспериментом как вообще сравнение LLMки и человека, и мне кажется очень прикольно было бы обучить модели на разное количество гиперпараметров и сравнить это с разными

процентилями результатов ЕГЭ (сначала только на второй задаче, но потенциально как я написал в предыдущем пункте можно будет это расширить на другие задачи ЕГЭ) и потом сделать график, типа сколько нам нужно гиперпараметров у LLMки чтобы обогнать результат школьников в РФ и там будет сколько гиперпараметров чтобы обогнать 50й, 75й, 99й персентиль результатов школьников в РФ.