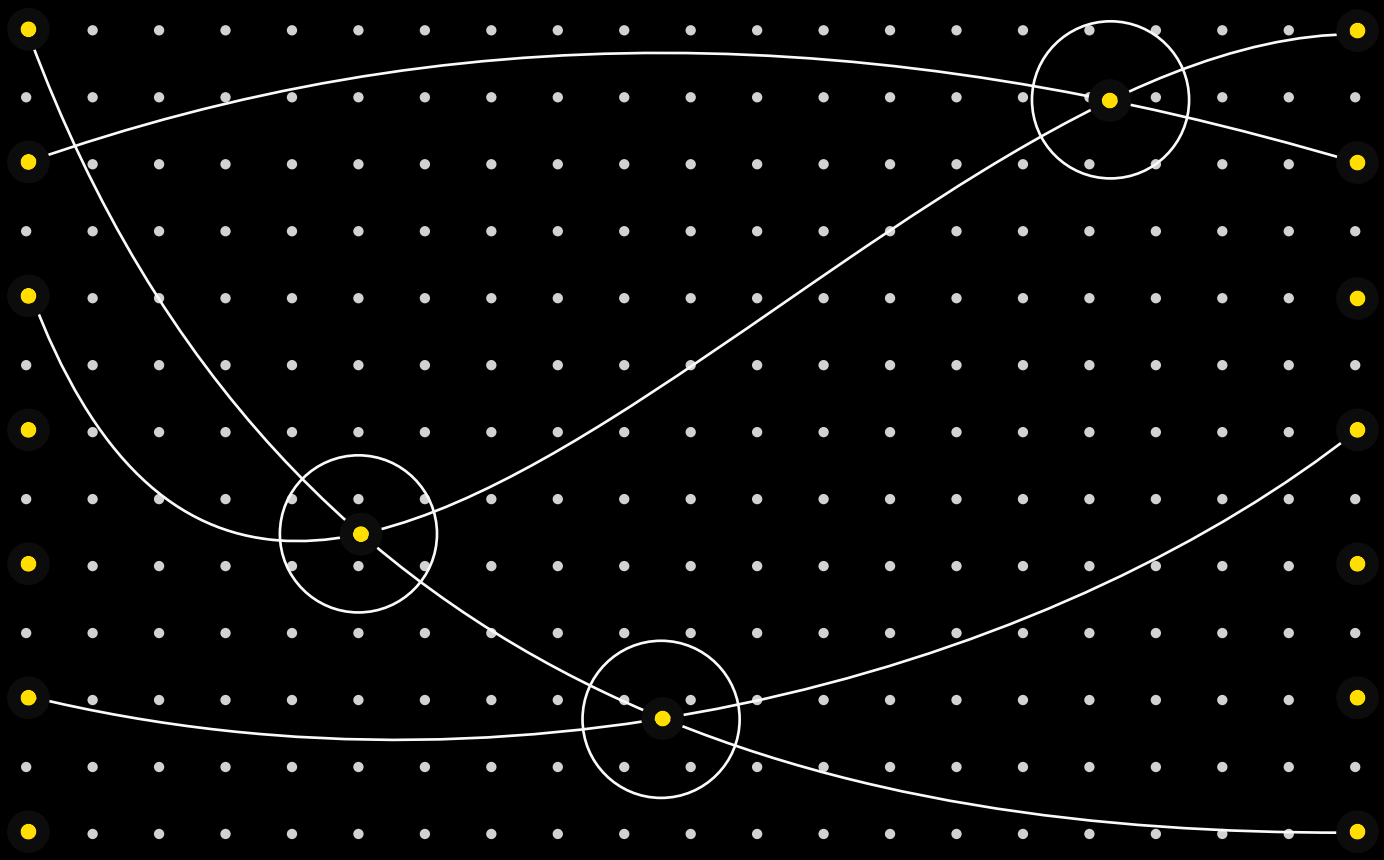
ЦЕНТРАЛЬНЫЙ УНИВЕРСИТЕТ

RL envs test task

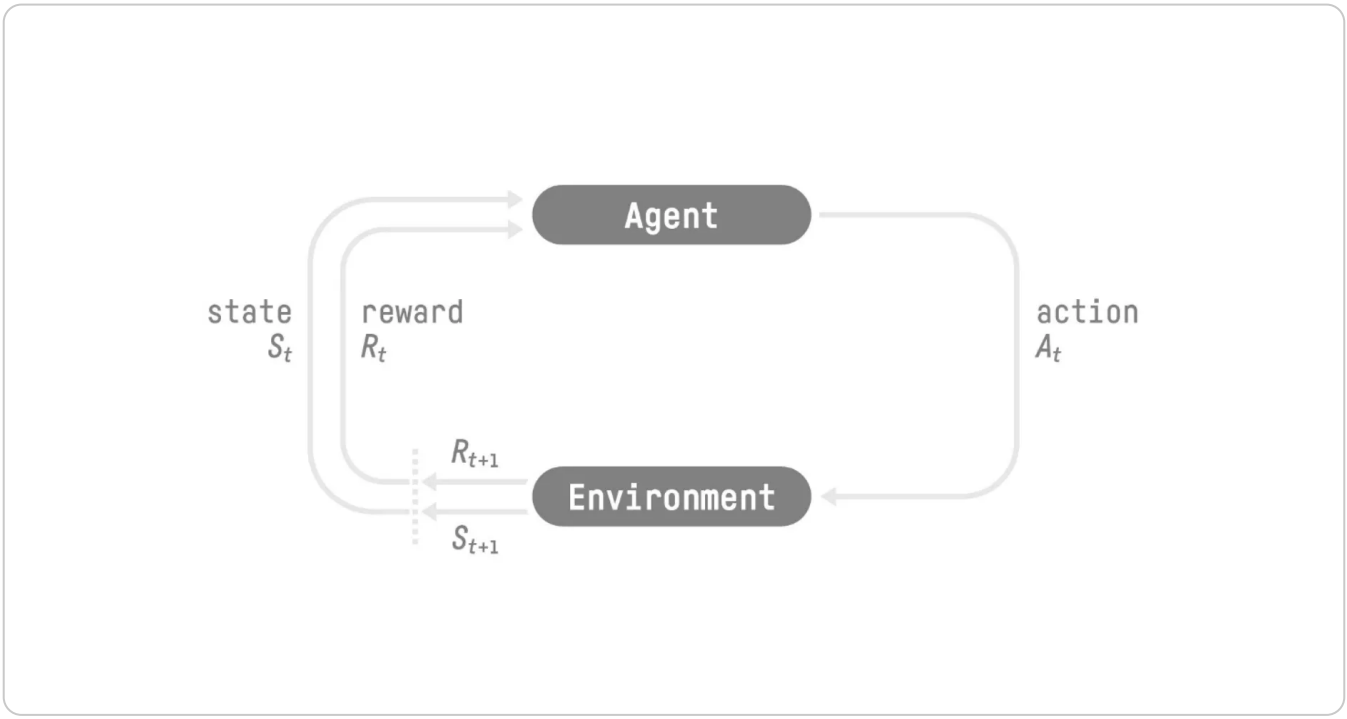


ИЗБРАННЫЕ ТЕМЫ ИССЛЕДОВАНИЙ В AI

ДОМАШНЕЕ ЗАДАНИЕ

НЕДЕЛЯ 2

Ваша задача обучить LLM агента, который будет взаимодействовать со средой. Обучать агента мы будем с помощью алгоритмов Reinforcement Learning.



The RL Process: a loop of state, action, reward and next state

В рамках данной схемы, наш агент получает от среды задачу (state). Решает задачу, посылает решение в среду (action). Получает за решение награду (reward). Каждый раз новый стейт — новая задача.

Среда

Сначала вам нужно придумать задачу для вашего LLM агента. Это должна быть задача, которая удовлетворяет следующим условиям:

- → ответ должен быть верифицируемым
- → сложность задачи должна быть регулируемой
- → для решения задачи не требуется многошагового взаимодействия со средой (задача решается за один запрос в LLM).

Пример среды — задача Binario. Цель игры, имея начальное состояние поля (в некоторых ячейках расставлены 0 и 1), дозаполнить его таким образом, чтобы в каждой строчке и в каждом столбце было одинаковое количество 0 и 1, при этом не было одинаковых строк и столбцов.

Данная задача верифицируется проверкой соответствия вышеуказанным правилам, сложность задачи можно регулировать меняя размер поля, LLM может заполнить все ячейки за один шаг (ей не нужна дополнительная информация).

В первой части задания ваша задача придумать и реализовать среду (ЗАПРЕЩЕНО брать среды из этого списка). Ваш класс должен наследоваться от класса `Env`.

```python
from abc import ABC, abstractmethod
from base.verifier import Verifier
from base.data import Data


class Env(ABC):
    """
    Base class for game
    @param name: name of the game
    @param verifier: class of the verifier
    """
    def __init__(self, name: str, verifier: Verifier):
        self.name = name
        self.verifier = verifier()

    @abstractmethod
    def generate(self, num_of_questions: int = 100, max_attempts:
int = 100, difficulty: Optional[int] = 1):
        """
        Generate game questions and answers
        @param num_of_questions: int
               @param max_attempts: int
        @return: list of Data
        """
        raise NotImplementedError("Game.generate() is not
implemented")

    def verify(self, data: Data, test_solution: str):
        """
        Verify whether the test solution is consistent with the
answer of the game data
        @param data: Data
        @param test_solution: str
        @return: bool
        """
        return self.verifier.verify(data, test_solution)

    @abstractmethod
    def extract_answer(self, test_solution: str):
        """
        Extract the answer from the test solution
        @param test_solution: str
        @return: str
        """
        raise NotImplementedError("Game.extract_answer() is not
implemented")
```

И реализовывать методы `generate` , `extract_answer` и `Verifier.verify`. Условие задачи и ответ от агента — это текстовые данные.

generate

- → num_of_questions — количество задач, которое должен вернуть метод.
- → max_attempts — количство попыток сгенерировать одну задачу. Скорее всего вы будете случайно сэмплировать условия задачи в цикле, но не любые случайно засэмплированные данные удовлетворяют семантике задачи, поэтому этот параметр отвечает за количество попыток сгенерировать одно условие. Пример с sudoku.
- → difficulty — сложность задачи, целое число в отрезке [1, 10]. Любая задача имеет набор гиперпараметров, которые регулируют её сложность. Например, для судоку — это количество ячеек с числами (чем их больше, тем легче). Вам необходимо придумать соответствие уровня сложности и гиперпараметров задачи. Пример с sudoku.
- → **Return** список из Data.

```
class Data:
    """
    Data class for game/corpus
    @param question: question of the game/corpus
    @param answer: answer of the game/corpus
    @param difficulty: difficulty of the game/corpus, from 1 to 10
    """
    def __init__(self, question: str, answer: str, difficulty: int = 1,
metadata: dict = None, **kwargs):
        self.question = question
        self.answer = answer
        self.difficulty = difficulty
        self.metadata = metadata
        self.gpt_response = ""

    def to_json(self):
        return {
            "question": self.question,
            "answer": self.answer,
            "difficulty": self.difficulty,
            "metadata": self.metadata,
            "gpt_response": self.gpt_response
        }
```

```python
    def to_json_str(self):
        return json.dumps(self.to_json(), ensure_ascii=False)

    @classmethod
    def from_json_str(cls, json_str):
        json_data = json.loads(json_str)
        return cls(**json_data)

    @classmethod
    def from_json_dict(cls, json_dict):
        instance = cls(**json_dict)
        if 'gpt_response' in json_dict:
            instance.gpt_response = json_dict['gpt_response']
        return instance

    @classmethod
    def from_jsonl_file(cls, file_path):
        data_list = []
        with open(file_path, "r") as f:
            for line in f:
                json_data = json.loads(line)
                instance = cls(**json_data)
                if 'gpt_response' in json_data:
                    instance.gpt_response = json_data['gpt_response']
                data_list.append(instance)
        return data_list
```

ВАЖНО

Важно, чтобы ваш метод generate принимал не только difficulty и преобразовывал их в набор гиперпараметров задачи, но и мог принимать набор гиперпараметров напрямую. Например здесь не передаётся difficulty, но передаются гиперпараметры. Ваша реализация должна поддерживать оба способа влиять на сложность задачи.

Строка question в Data должна содержать промпт с условиями задачи. Для этого создайте отдельный файлик с функцией, которая к конкретной конфигурации задачи добавляет правила игры **на английском языке**. Пример. Отнеситесь к этому пункту ответственно. Правила игры должны интерпретироваться однозначно и покрывать все возможные случаи.

verify

Среда должна уметь не только генерировать условие задачи, но и проверять решение, полученное от LLM агента. Для этого существует этот метод, вам **не нужно** его переопределять, но вам нужно реализовать интерфейс `Verifier` и передать его в конструктор класса `Env`.

```
class Verifier(ABC):
    """
    Base class for verifier
    """
    def __init__(self):
        pass

    @abstractmethod
    def verify(self, data: Data, test_answer: str):
        """
        Verify whether the test answer is consistent
with the gold answer
        @param data: Data
        @param test_answer: str
        @return: bool
        """
        raise NotImplementedError("Verifier.verify()
is not implemented")

    @abstractmethod
    def extract_answer(self, test_solution: str):
        """
        Extract the answer from the test solution
        @param test_solution: str
        @return: str
        """
        raise NotImplementedError("Verifier.extract_answer() is not
implemented")
```

Метод verify должен извлечь ответ из `test_answer` методом `extract_answer` и проверить корректность сравнением с `data.answer`. Так как корректность не всегда проверяется простым сравнением, логика метода `verify` может быть сильно сложнее. См примеры в библиотеке SynLogic.

extract_answer (одинаковый в Verifier и Game)

- test_solution: str — генерация LLM. Так как генерация может содержать помимо ответа, цепочку рассуждений, необходимо корректно извлекать финальный ответ из того, что выдаёт LLM.
- **Return:** str — ответ на задачу.

Посмотрите примеры в библиотеке SynLogic.

LLM агент

- Насэмплируйте датасет из задачек для обучения вашего агента.
- Создайте несколько обычных датасетов с данными для тестирования модели. Каждый датасет содержит данные своей сложности. В этих датасетах задачи НЕ должны сэмплироваться (сгенерированы один раз и не меняются), это нужно для воспроизводимости результатов.
- Напишите код обучения агента аналогично тому, как это сделано в этом ноутбуке. Для обучения предлагается использовать Qwen2.5-1.5b-Instruct.
- Используйте unsloth для более быстрого обучения. Оберните вашу модель в FastLanguageModel.
- Не забудьте добавить system prompt

```
SYSTEM_PROMPT = """
Respond in the following format:
<think>
...
</think>
<answer>
...
</answer>

"""
```

→ Ваш GRPO Trainer будет выглядеть следующим образом.

```
trainer = GRPOTrainer(
    model = model,
    processing_class = tokenizer,
    reward_funcs = [
        correctness_reward_func,
    ],
    args = training_args,
    train_dataset = dataset,
)
trainer.train()
```

где `dataset` — это iterable dataset, который вы создали в первом пункте,
а `correctness_reward_func` — это обёртка над `Game.verify`

→ Опишите как изменилась точность обученной модели по сравнению с Qwen2.5-1.5b-Instruct (не забывайте SYSTEM_PROMPT при измерении метрик, используте vllm для инференса). Нарисуйте парные бары (для двух моделей) для датасетов разной сложности подобным образом: https://matplotlib.org/stable/gallery/lines_bars_and_markers/barchart.html#sphx-glr-gallery-lines-bars-and-markers-barchart-py

→ Сохраните обученную модель и датасеты, на которых вы измеряли качество, на huggingface.

Правила

- Проанализируйте полученные результаты. Это **самый важный пункт**, потому что хочется увидеть не только числа с полученными метриками. Как вы объясняете увиденное поведение? Напишите **отчет** о проведенных экспериментах. Что получилось? Что нет?
- **Нет правильного способа решить задачу**. Не стоит беспокоиться, что вы делаете что-то неправильно. Мы хотим увидеть ваши способности к исследованиям, а не какое-то конкретное решение задачи. Возможно, вы придумаете то, о чем мы даже не задумывались изначально — это будет высший класс.
- Следуйте интерфейсам, описанным в условиях задачи.
- Вы можете использовать Google Colab или Kaggle Code, чтобы получить доступ к бесплатным вычислительным ресурсам.
- Присылайте решение в виде репозитория на github с отчетом по решению и чёткими инструкциями, как запустить ваш код. **Убедитесь, что мы сможем запустить ваше решение по этим инструкциям**.
- Вы можете использовать любые библиотеки и фреймворки, которые вам могут быть необходимы.
- **Сфокусируйтесь на том, чтобы код был чист и понятен**. Если вы считаете, что какая-то его часть может быть непонятна, то добавьте комментарии. Мы очень сильно ценим хорошо написанный код, поэтому **если решение задачи будет оформлено грязно, то мы можем его отклонить**.